O'REILLY

TURING 图灵程序设计丛书

Arduino技术指南

Arduino: A Technical Reference

Arduino爱好者案头常备

了解电子产品设计思路，实现创客梦想



[美] John M. Hughes 著

武传海 译

中国工信出版集团 人民邮电出版社 POSTS & TELECOM PRESS

数字版权声明

图灵社区的电子书没有采用专有客户端，您可以在任意设备上，用自己喜欢的浏览器和PDF阅读器进行阅读。

但您购买的电子书仅供您个人使用，未经授权，不得进行传播。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。

图灵推荐阅读

无需专业背景也能看懂的电子技术书

《电子工程师必读：元器件与技术》

《物联网设计：从原型到产品》

《爱上电子学：创客的趣味电子实验（第2版）》

《电子电气工程师必知必会》

无需专业背景也能上手的编程入门书

《Python编程：从入门到实践》

《写给大家看的安卓应用开发书：App Inventor 2快速入门与实战》

数字时代生存指导书

《用户思维+》

《用数据讲故事》

《写给大家看的设计书（第4版）》

《番茄工作法图解》

《单核工作法图解》

图灵程序设计丛书

Arduino技术指南

Arduino A Technical Reference

[美] John M. Hughes 著
武传海 译

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

O'Reilly Media, Inc.授权人民邮电出版社出版

人民邮电出版社
北京

图书在版编目（CIP）数据

Arduino技术指南 / （美）约翰·休斯
(John M. Hughes) 著 ; 武传海译. -- 北京 : 人民邮电
出版社, 2017.12
（图灵程序设计丛书）
ISBN 978-7-115-47105-5

Ⅰ. ①A… Ⅱ. ①约… ②武… Ⅲ. ①单片微型计算机
－程序设计－指南 Ⅳ. ①TP368.1-62

中国版本图书馆CIP数据核字(2017)第260883号

内 容 提 要

本书主要讲解了Arduino开发板的物理特性与接口功能，Arduino使用的各种AVR微控制器，Arduino特有的编程环境，各种扩展板，可与Arduino一起工作的传感器、继电器模块、小键盘以及其他附加组件，从零开始创建自定义扩展板的步骤，介绍各种分析设计问题、定义实体与可测试需求的方法，确保开发成功。

◆ 著　　[美] John M. Hughes
译　　武传海
责任编辑　陈　曦
责任印制　彭志环

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　　　　印刷

◆ 开本：800×1000　1/16
印张：32
字数：756千字　　　　2017年12月第1版
印数：1－3 000册　　　2017年12月北京第1次印刷
著作权合同登记号　图字：01-2017-6479号

定价：129.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

版权声明

O’Reilly Media, Inc.介绍

O’Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O’Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O’Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O’Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 *Make* 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O’Reilly 的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O’Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过图书出版、在线服务或者面授课程，每一项 O’Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O’Reilly Radar 博客有口皆碑。”

——*Wired*

“O’Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——*Business 2.0*

“O’Reilly Conference 是聚集关键思想领袖的绝对典范。”

——*CRN*

“一本 O’Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——*Irish Times*

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野，并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去，Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——*Linux Journal*

目录

前言

自 2005 年推出以来，Arduino 已经成为世界上最成功（有些人可能对此持有异议）的开源硬件项目之一。Arduino 团队公开其设计之后，包括意大利、巴西、中国、荷兰、印度和美国在内的很多国家，都在生产基于这种设计的开发板。无论是谁，只要花大约 15 美元，就能买到一块功能齐全的 Arduino 兼容开发板。人们可以自由下载并使用完全免费的 Arduino 开发环境。最初，Arduino 采用 8 位 AVR 系列微控制器（AVR 本身就是一款有趣的设备，其历史也同样有趣），后来进入 32 位处理领域，相关产品包括采用 ARM 处理器的 Due、带有板载模块运行 OpenWrt（一种 Linux 发行版本）的 Yún，以及即将推出的 Zero。Arduino 开发板广泛应用于各种项目，从互动艺术到机器人，从环境监测传感器到 CubeSat 卫星（由小团队制造，发射费用只相当于常规卫星的一小部分）中的智能组件，都能看到它的身影。

许多年前，我购买了生平第一块 Arduino 开发板（Arduino Duemilanove），这更多是出于好奇，而不是别的什么原因。从 20 世纪 80 年代早期开始，我就与微处理器和微控制器开发系统打交道，先是 6502、6800 和 8051，然后是 8086、Z80、80186、68000 系列。早先，我一般使用汇编语言或 PL/M 为这些设备编写程序，对于嵌入式系统，这些语言是当时仅有的合理选择。后来开始使用 C 或 Ada 语言，这得益于微处理器性能的提升以及软件工具的成熟。不管怎样，我都希望有很多资料可供参考，如数据手册、说明书、参考手册以及设计文档。这些资料往往随开发电路板及其配件一起提供，常常会装满一个又大又重的盒子。

当购买的 Arduino 到手时，我发现只有一个很小的盒子，里面只有一块电路板、一个插入式电源组、几个 LED 灯、一些电阻、跳线，以及一块免焊面包板。里面没有说明书，没有参考手册，没有数据手册，也没有包含文档与软件的 CD 光盘，只有几页纸，上面列出了包装清单，以及一个 URL 网页地址。我进入 Web 页面，读了一些“入门”资料，并找到所需软件的链接。毫不夸张地说，我当时真的很惊讶。

那时，我对 Arduino 了解得并不多。购买 Arduino 时，我根本不知道它背后的故事，也不了解它的目标用户。后来我才知道，它主要针对有很少或根本没有技术背景的用户。这些用户只想尝试做一些很酷的玩意儿（纯粹为了好玩），并让它们运行起来。换言之，Arduino 面向的是艺术家和热衷于捣鼓东西的人，而不是那些对技术细节感兴趣以及痴迷于项目规划、技术规范与操作说明书的工程师。

当我明白这一点后，一切就明了起来。后来，我拜读了 Massimo Banzi 先生写的《爱上 Arduino》一书，它让我更好地理解了 Arduino 所信奉的哲学。在我探寻更多细节的过程中，这本书是一个很好的起点。与那些拥有自身开发套件的半导体制造商不同，Arduino 团队并不销售芯片，而是一直致力于激发人们的创造力。他们之所以选用 AVR 微控制器，是因为它价格便宜，并且很适合用来实现自己的想法，让这款设备能够轻松应用到创意工作中。AVR 拥有很强大的计算能力，并且内置了充足的内存，可以做一些复杂而有趣的工作，这是早期微控制器所不具备的。早期微控制器往往需要昂贵的开发工具，提供的片上内存也十分有限。

除了简单与便宜外，Arduino 获得成功的真正秘诀在于，AVR 芯片中的 BootLoader 固件、简单易用的集成开发环境，以及一起提供的各种代码库（在开源与 Creative Commons 许可下，用户可以免费使用）。Arduino 信奉的哲学是，让用户使用 Arduino 的过程变得尽可能简单。通过隐藏大量技术细节和简化开发流程，Arduino 鼓励用户大胆实验，尝试各种新想法，以及尽情玩乐。长时间以来，我第一次发现自己很喜欢以不同组合连接各种东西，并思考自己能用它做什么。我曾经讲过嵌入式系统设计课程，那时 Arduino 开发板尚未问世，真希望那个时候就有 Arduino，它能大大减少学生学习汇编语言、内存映射、流程图等内容时产生的挫败感。

收到第一块 Arduino 开发板之后，我找到了许多有用的资源，还有大量有趣的 Arduino 附加组件，其中一些在价格与功能上很让人满意 [1]。从此，我喜欢上了收集各类 Arduino 小玩意儿，购买便宜的扩展板和模块，根据想法搭建各种小设备。但有一点很让我伤心，当我打开装有漂亮“小玩意儿”的盒子时，经常发现里面没有任何说明文档，甚至连电路图也没有。

本来我满怀欣喜地想买些有趣的东西，最后却发现它根本没有说明文档。作为工程师，我对于这点感到特别沮丧，然后不得不自己去查找，看看到底有没有我能读的文档（我不懂中文）。有时这种查找一无所获，我不得不根据相关数据手册对电路板进行逆向分析，以便搞清楚它的电路设计。还有些时候，我的确能够找到相关资料，但这些资料往往比较零碎，散布在多个网站的不同地方。虽然这种情况正慢慢好转，但仍然让人十分头疼。过去数年间，我收集了大量笔记、网页链接和数据手册，最后下定决心把它们好好整理一下，放到同一个地方。

那么，本书包含了哪些你在网上找不到的内容？坦率地说，这样的内容并不多，但我真心希望本书能够帮你减少很多挫折并节省时间。当然，书中也包含了许多我自己发现的内容。官方技术数据来自相关产品的制造商，比如 Atmel、Arduino 团队，以及其他很多厂商。Atmel 和 Arduino 团队为人熟知，却又非常低调。一些国外厂商至少会有像样的网站，有些网页甚至十分漂亮，其中包含相关技术文档的链接。本书囊括所有我能找到或者通过逆向分析得来的基本数据，在我力所能及的范围之内，尽量保证这些数据准确无误。我不

注 1：在 eBay 网的搜索框中输入 Arduino，将会返回大量搜索结果，比如超声波距离传感器、温湿度传感器、各种 Arduino 克隆板、蓝牙与 ZigBee 扩展板，等等。但令人遗憾的是，有些电子元器件只有少量文档，甚至没有文档；或者即使有一些说明文档也大都是旧的，内容不准确。当然，这并不意味着你不应该考虑这些卖家（他们开出的价格往往很诱人，并且电子元器件的质量也非常好），但最好还是认真考虑一下这方面的问题，网络购物遵循的一般规则是“一旦售出，概不退换”。

想让你经历我所遇到的挫折，例如那些 USB 接口的琐碎的技术细节，或者搞清楚扩展板无法正常工作的原因，或者了解你从 eBay 购买的传感器为什么不工作。

我所经历的一系列挫折最终促成了本书的诞生，在学习使用 Arduino 开发板与扩展板的过程中，我一直想有这样一本书用来参考。在工作台上，我真的需要有这样一本实体书放在手边，可以随时查阅。因为上网查阅资料有时是不可行的，甚至有时根本上不了网（比如在山顶上试图为公司调试一个远程数据记录设备时，手上只有一台小小的上网本，方圆 100 km 没有任何无线信号）。我希望手上有这样一本书，不管身在何处都可以快速查阅，并解决 Arduino 及相关附件使用过程中遇到的问题。希望你会觉得这本书有用，就像我整理本书笔记时感受到的那样。

目标读者

本书面向那些需要知道或想知道 Arduino 技术细节的读者。也许你已经尽力了解了相关介绍材料，并阅读了大量诸如“超赞项目 99”的图书。而现在，你需要知道如何做出一些新颖独特的玩意儿。或者，你是一个工程师或研究员，想把 Arduino 集成到你实验室的实验装置中。你甚至有可能是遥控飞机玩家，想把 Arduino 集成到模型飞机中，又或者想把它集成到 DIY 气象站[2]中，或者用它来做一些更具野心的东西（比如 CubeSat）。

理想情况下，你应该具备基本的 C 或 C++ 知识，了解电子如何在一个电路中运动，拥有一些搭建电子设备的经验。容我斗胆进言，建议你也要购买我的《电子工程师必读：元器件与技术》放在手边，同时准备一些关于编程与电子元器件的参考资料（更多参考资料详见附录 D）。

这本书是什么

本书是一本参考书，也是一本技术手册。在内容上我精心编排，以便你能够轻松、快速地查到要找的内容。介绍相关内容时，我尽量给出信息来源。本书包含了许多我的个人见解，它们是我多年研究所得，希望对你有用。

这本书不是什么

本书不是一本学习教程，这也不是我的主要目标。书中并未讲解基本的电子电路知识，也没有讨论如何使用 C++ 语言为 Arduino 编写程序。你可以通过各种渠道找到很多专门讲解电子电路与编程知识的教程，我希望各位可以把本书作为学习这些内容的起点。

本书也不是 Arduino 官方认可的各种 Arduino 产品介绍指南。书中依据的信息来源于各种渠道，其中有些内容比较易懂，有些则比较晦涩，当然，书中也包含我自己的一些笔记与注解，它们都是我使用 Arduino 的经验总结。因此，我将对书中出现的任何错误与疏漏负责。

注 2：*Environmental Monitoring with Arduino and Atmospheric Monitoring with Arduino* 一书由 Emily Gertz 与 Patrick Di Justo 合著，介绍了如何使用便宜且容易获取的传感器与 Arduino 制作环境监测设备，并提出了很多想法与建议。

专业术语

20 世纪 80 年代早期，人们首次把处理器、微处理器、微控制器区分开来。当时，各大制造商正试图通过大小与外部电路（设备用它们来做一些有用的事）的数量区分各自的产品。通常，大型机处理器与更小的微处理器（比如桌面 PC 中使用的那些）都需要一些外部组件（有时需要很多）才能使用。另一方面，微控制器已经内置了它工作时需要的所有部件。微处理器通常都支持外部存储器，而对于添加外存以扩展芯片存储这一方面，微控制器提供的支持很有限（虽然十分有限，但毕竟还是支持的）。

整本书中，“微控制器”与“处理器”两个术语是通用的。尽管“微控制器”这一称呼更多着眼于技术方面，但在我看来，它仍旧是一个数据处理器，只不过是一个小版本的“大型机”（很久以前，我曾经用过大型机）。其实，它们所做的工作都是一样的，只是大小与处理速度不同而已。

本书内容

第 1 章简单介绍各种 Arduino 开发板的历史；还介绍 Arduino 开发板中使用的 AVR 微控制器，讨论 Arduino 软件兼容产品与硬件兼容产品的不同。

第 2 章的主题是 Atmel AVR 微控制器，概述了一个真正复杂的设备是如何构成的。当然，这里只对最重要的部分做快速浏览，包括定时器逻辑、模拟比较器、模拟输入、SPI 接口，以及芯片上其他主要子系统。

第 3 章进一步讲解 Arduino 开发板上使用的各种 AVR 微控制器，包括 ATmega168/328、ATmega1280/2560、ATmega32U4。这一章内容以第 2 章内容为基础，添加了更多底层细节，比如内部架构、电气特性、芯片引脚布局等。

第 4 章讲解各种 Arduino 开发板的物理特性与接口功能，包括 USB 接口类型、印制电路板（PCB）大小、开发板引脚布局图等。

第 5 章讲解 Arduino 编程环境，这是真正使其与众不同的地方。这一章还介绍 Arduino 程序的定义，以及如何使用 C 与 C++ 语言编写 Arduino 程序；同时，还介绍 Arduino BootLoader 与 `main()` 函数，讲解如何下载 Arduino 源代码。通过阅读这些源代码，你可以了解隐藏在其表面之下的底层工作原理。

第 6 章介绍 AVR-GCC 工具链，以及在不使用 Arduino IDE 的前提下，为 Arduino 开发板编程的技术。其中，还涉及与“生成文件”（makefiles）有关的内容，并简单介绍汇编语言编程。最后介绍将代码上传到 AVR 的各种工具。

第 7 章的讲解重点是 Arduino IDE 自带的各种标准库。Arduino IDE 本身提供了大量标准库，并且不断添加更多库。如果想了解某个特定传感器或特定操作是否有现成的库模块，那么这一章是个不错的起点。

第 8 章介绍适用于 Arduino 的各种扩展板，包括各种常见类型，比如闪存、原型、输入 / 输出、以太网、蓝牙、ZigBee、伺服控制、步进电机控制、LED 显示屏、LCD 显示器等。

这一章也包含使用多重扩展板的内容，还提供许多提示与技巧，帮你最大限度地挖掘扩展板的潜力。

第 9 章介绍一些可以与 Arduino 开发板配合使用的附加组件，包括各种传感器、继电器模块、小键盘，以及其他一些非特定于 Arduino 但能很好与其配合工作的器件。此外，还提供许多所讨论的电子元器件的引脚布局与电路图。

第 10 章专门介绍如何自己动手制作扩展板，因为找到满足我们需要的扩展板有时并非易事。此外，还讲解如何在没有 Arduino 类型电路板的情形下使用 AVR 微控制器，并且仍能使用 Arduino IDE。

第 11~13 章介绍几个电子设计项目，通过这些项目可以进一步了解 AVR 微控制器与 Arduino 扩展板的功能。这些电子项目也展示了如何将 Arduino 应用于各种情形，而并非只演示如何制作电路板或设备。当然，只要愿意，你完全可以自己动手制作这些电子项目，并以此为起点制作自己的项目。介绍每个示例项目时，都包括工作原理、电路图、详细元件列表、PCB 布局设计（若需要），以及运行所需软件的概述。

由于本书的讲解重点是 Arduino 硬件与相关模块、传感器、组件，所以说到“软件”时，只讲述软件最重要的部分。请注意，书中给出的代码都是不完整的，不能直接运行。关于示例项目的完整软件，可以在 GitHub（https://www.github.com/ardnut）上找到。

第 11 章介绍如何制作一个基本的信号发生器，测试电子电路中会经常用到它。通过这个信号发生器，你可以产生具有各种占空比的脉冲，输出一个系列脉冲响应触发脉冲输入，生成正弦波，也可以产生可编程脉冲模式。

第 12 章讲解如何设计与制作一个智能调温器，它很适合与家中的 HVAC（加热、通风、空调）系统一起使用。学过这一章就不用再购买现成的调温器了，因为你完全可以自己动手制作一个，并让它完全根据你的设想工作。在这一章中，我不仅教你如何集成温度传感器（特点是包含多个温度与湿度传感器），还会教你如何使用 HVAC 系统的风机营造一个舒适的环境，并且不会有因运行压缩机或点亮加热器而产生的费用。

第 13 章将了解如何制作一个自动的模型火箭发射器，它带有可编程定序器与自动系统检测功能。即使手上没有模型火箭，也建议你认真学习本项目中使用的一些技术，因为这些技术可以用在多种对执行顺序有严格要求的控制过程中，比如工厂里的生产线、实验室中的自动材料搬运设备。

附录 A 简单介绍一些常用的工具与附件，如果不想使用现成的电路板与模块，那么可能需要使用这些工具自己动手制作。

附录 B 总结微控制器的控制寄存器，涉及的微控制器有 ATmega168/328、ATmega1280/2560 以及 ATmega32U4。

附录 C 列出一些销售 Arduino 以及相关兼容产品的经销商与厂商，这些名单并不全面，但足够供你用作探索起点，找到自己需要的电子元器件。

附录 D 是我推荐阅读的一些图书，其中不仅包括讲解 Arduino 的图书，还包括讲解 C 与 C++ 编程以及电子元器件的图书。

附录 E 对一些常用的 Arduino 与 AVR 软件开发工具进行概述，它们目前都是可用的，你可以很容易地获取。

产品宣传

书中提到了 Arduino 团队及其官网 Arduino.cc，此外没有对任何其他产品做宣传，即使有也不是有意为之。在本书讲解中，我提到了许多不同的元件制造商、供应商，以及其他作者。我尽量对他们一视同仁，不偏不倚。之所以提到他们完全是因为我恰好有他们的产品，并且已经在一些电子项目（包括一些演示项目）中成功使用了他们生产的扩展板、模块、传感器或 Arduino PCB（或 PCB 克隆板）。书中涉及的任何商标均由各所有者持有，列出它们仅仅是为了方便各位参考。关于书中图片，我尽量使用自己的元件、工具、电路板、模块以及其他项目，有些产品图片中给出了元件型号及相关厂商，但这不意味着它是唯一可用的，只是因为我恰好有它，并在电子制作中使用了。某些情况下，我会使用那些带有相关厂商与创建者许可的图片、公共领域的作品，或者拥有 CC 许可的图片，书中都做了相应的标注与说明。书中所有框图、电路图以及其他插图（不包括照片）都是我自己制作的，对于其中的任何错误或疏漏，我负全部责任。

排版约定

本书使用以下排版约定。

- 楷体
  表示新术语和重点强调的内容。
- 等宽字体（`constant width`）
  表示程序片段，以及正文中出现的变量、函数名、数据库、数据类型、环境变量、语句和关键词等。
- 加粗等宽字体（**`constant width bold`**）
  表示命令以及其他需要用户输入的文字。
- 等宽斜体（*`constant width italic`*）
  表示这些值应该替换为用户输入，或根据上下文确定。



该图标表示提示或建议。



该图标表示一般性说明。



该图标表示警告或警示。

Safari ®Books Online

Safari Books Online 是应运而生的数字图书馆，它同时以图书和视频的形式出版世界顶级技术和商业作家的专业作品。

技术专家、软件开发人员、Web 设计师、商务人士和创新专家等，都将 Safari Books Online 作为开展调研、解决问题、学习和认证培训的首选资源。

Safari Books Online 为企业、政府、教育和个人提供各种产品组合和灵活的定价策略。

会员可以通过搜索，从数据库中访问数以千计的图书、培训视频和正式出版前的书稿，这些数据来源包括 O'Reilly Media、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit Press、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kauf-mann、IBM Redbooks、Packt、Adobe Press、FT Press、Apress、Manning、New Riders、McGraw-Hill、Jones & Bartlett、Course Technology 等。要了解 Safari Books Online 的更多信息，请访问我们的网站（http://www.safaribooksonline.com/）。

联系我们

请把对本书的评价和问题发给出版社。

美国：

> O'Reilly Media, Inc.
> 1005 Gravenstein Highway North
> Sebastopol, CA 95472

中国：

> 北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）
> 奥莱利技术咨询（北京）有限公司

我们为本书提供了专门网页，上面有勘误表、示例以及其他信息。可以通过 http://bit.ly/hadoop-security 访问该网页。本书中文版勘误可到 http://www.ituring.com.cn/book/1866 提交。

为本书提供建议或咨询技术问题，请发邮件到 bookquestions@oreilly.com。

想了解更多关于 O'Reilly 图书、培训课程、会议和新闻的信息，请访问以下网站：
http://www.oreilly.com。

我们的其他联系方式如下：
Facebook：http://facebook.com/oreilly

Twitter：http://twitter.com/oreillymedia
YouTube：http://www.youtube.com/oreillymedia

致谢

尽管写作看起来是一种习惯使然，但如果缺少了家人长久的支持与耐心，这本书也是不可能问世的。写作期间，他们一直在身边鼓励我、支持我，甚至给我带吃的，还时不时闯到我办公室，看看我是不是还活着，真的没有比他们更好的家人了。尤其感谢我的女儿 Seren，她帮我拍了许多照片，还帮忙分类并整理了我收集的各种东西。

还要感谢 O'Reilly 出版公司的编辑们，感谢他们再次给我合作机会。他们一如既往地积极给予帮助，对我一直保持着耐心，并表现出极大的宽容。最后，特别感谢 Brian Sawyer 与 Dawn Schanafelt，他们提供了很棒的编辑支持与指导。还有 Mike Westerfield，感谢他做了深入的技术评审工作。

电子书

扫描如下二维码，可购买本书电子版。

第 1 章

Arduino家族

本章简单介绍 Arduino 的历史，主要指 2007 年以来推出的各种开发板的简短家谱，并不涵盖 2007 年以前生产的开发板，也无意囊括其各种仿制品与衍生物。本章重点在于介绍几种主要的 Arduino 开发板的不同之处，其中特别关注的是开发板使用的处理器类型以及物理设计。此外，本章也将简单介绍 Arduino 电路板的应用领域。

第 2 章介绍有关 Atmel AVR 处理器内部功能的基本信息。第 3 章介绍 Arduino 开发板上使用的特定处理器。第 4 章讲解不同类型的官方 Arduino 开发板（本章内容）的物理特性，但并不包括 Arduino Yún 开发板。

1.1 Arduino简史

2005 年，在意大利伊夫雷亚交互设计研究所（Interaction Design Institute Ivrea），Massimo Banzi 与 David Cuartielles 基于 Hernando Barragán（Wiring 发明者）的工作创造了 Arduino。它是一种易用的可编程设备，设计初衷是用于交互艺术设计项目。随后，David Mellis 基于 Wiring 开发了 Arduino 软件。不久之后，Gianluca Martino 与 Tom Igoe 加入该项目，他们 5 人被公认为 Arduino 的创始人。Arduino 的诞生源于他们想开发这样一种简单设备：这种设备应当能够很容易地连接到其他各种设备上，比如继电器、发动机、传感器，也应当很容易进行编程；并且价格也要便宜，毕竟学生和艺术家没多少闲钱。最后，他们选择了 Atmel 公司生产的 8 位微控制器（又称 MCU 或 μC）设备中的 AVR 系列，设计了一种易于连接使用的独立电路板，并为微控制器编写了 Bootloader 固件，然后将之全部打包放入一个简单的集成开发环境，其中使用的程序称为 sketch。这最终促成了 Arduino 的诞生。

此后，Arduino 朝着几个不同的方向发展，其中一些版本变得更加小巧，而另一些版本则变得更大。每个版本的 Arduino 都有特定的市场定位，以满足不同用户的需求。这些不

同版本的 Arduino 的共同之处是，它们都提供 Arduino runtime AVR-GCC 库，并配备了 Arduino 开发环境，且都有板载 Bootloader 固件，该固件会被预先加载到每个 Arduino 开发板的微控制器。

Arduino 系列开发板使用的处理器由位于美国加利福尼亚州圣何塞市的 Atmel 公司开发。各版本 Arduino 的大部分设计安排都是围绕着使用 8 位 AVR 系列微控制器展开的，但 Arduino Due 除外，它搭载的是一块 ARM Cortex-M3 32 位处理器。本书不会介绍 Arduino Due，因为它在许多方面都与 AVR 设备完全不同。Arduino Due 和基于 ARM Cortex-M3 设计的类似微控制器真的值得我们单独讨论。

尽管正如 Arduino 团队所说，Arduino 开发板只是基本的 Atmel AVR 开发板，但它也配备了 Arduino 软件环境，这使其与众不同。这是所有 Arduino 用户共同的经验，也是 Arduino 理念的基石。第 5 章将介绍 Arduino IDE 和随 IDE 一起提供的库，以及 Bootloader。第 6 章讲解如何在不使用 Arduino IDE 的前提下为 AVR MCU 开发软件。

1.2 Arduino设备类型

这些年来，Arduino.cc 的设计师已经开发了多种 Arduino 开发板。首先得到广泛传播的是 Arduino Diecimila 开发板，它发布于 2007 年。自从第一个版本发布以来，Arduino 系列开发板已经演化为可以充分利用各类 Atmel AVR MCU 的设备。Arduino Due 开发板发布于 2012 年，它是第一个采用 32 位 ARM Cortex-M3 处理器的 Arduino 开发板。在处理性能与引脚配置方面，Arduino Due 大大强于其他 Arudino 开发板。另一些开发板，比如 LilyPad 和 Nano，也并不像其他 Arduino 家族成员一样拥有相同的引脚配置，而是针对不同的应用场景进行设计。比如 LilyPad 应用于可穿戴设备，Esplora 应用于手持设备，而 Mini、Micro 和 Nano 则适用于对便携性与大小有严格要求的紧凑型设备。

Arduino 每年都在不断推出新型开发板，当你阅读本书时，书中列出的开发板可能已经过时了。新型 Arduino 开发板会采用更强大的处理器，拥有更大内存，还配有性能大幅增强的输入 / 输出（I/O）功能。但是，它们大部分都会采用相同的引脚排列，并且都与现有的扩展板（shield）以及各种扩展组件（比如传感器、继电器、驱动器）协同工作。表 1-1 列出了 2007 年以来陆续推出的各种 Arduino 开发板。相比于较旧版本，较新版本的 Arduino 开发板可能在某些方面进行了微调，并带有更新的函数库。但它们仍能正常运行之前为较旧版本的开发板编写的大部分程序，反之则未必可行。

表 1-1 不是购买指南，它只用来让你对 Arduino 的历史有大致的了解。可以看到，2007 年与 2008 年陆续推出了 LilyPad、小型板（比如 Nano、Mini、Mini Pro）以及从 Diecimila 自然演化而来的 Duemilanove。尽管 Duemilanove 与 Diecimila 之间没有显著的外观差异，但 Duemilanove 对电源供给做了一些改进，尤其是在 USB 供电与外部 DC（直流电）供电之间实现了自动切换。Duemilanove 的后续版本也采用了 ATmega328 MCU，它为程序提供了更多内存。

表 1-1 并不包含 Arduino Robot，这是一种 PCB，并配有发动机与轮子。在 Arduino 产品阵容中，Yún 是最新的开发板之一。它是一个有趣的“家伙”，既带有一个 ATmega32U4 微控制器，又配有一块 Atheros AR9331 MIPS 处理器，该处理器用来运行一个叫作 Linino

（基于 OpenWRT 操作系统）的 Linux 发行版本。我并不打算深究 Yún 的 OpenWrt 端，但 Arduino 端基本上是一个标准的 Arduino（具体地说是 Arduino Leonardo）。如果你想学习更多有关 Yún 的内容，建议访问 Arduino 官方站点（https://www.arduino.cc/en/Main/ArduinoBoardYun）。

表1-1：Arduino产品时间线

产品名称	年份	微控制器	产品名称	年份	微控制器
Diecimila	2007	ATmega168V	Mega 2560	2010	ATmega2560
LilyPad	2007	ATmega168V/ ATmega328V	Uno	2010	ATmega328P
Nano	2008	ATmega328/ ATmega168	Ethernet	2011	ATmega328
Mini	2008	ATmega168	Mega ADK	2011	ATmega2560
MiniPro	2008	ATmega328	Leonardo	2012	ATmega32U4
Duemilanove	2008	ATmega168/ ATmega328	Esplora	2012	ATmega32U4
Mega	2009	ATmega1280	Micro	2012	ATmega32U4
Fio	2010	ATmega328P	Yún	2013	ATmega32U4 + Linino

表 1-1 中，若某款产品的“微控制器”一列显示多个控制器，则表明该特定版本的 Arduino 开发板起初采用的是第一个微控制器，后来又采用了另外一种控制器（通常功能更加强大）。比如较早版本的 Duemilanove 采用的是 ATmegal168，而较新版本的 Duemilanove 采用的则是 ATmega328。从功能上看，ATmegal168 与 ATmega328 完全相同，但 ATmega328 拥有更大内存。

Leonardo、Esplora、Micro、Yún 等 Arduino 系列最新产品均使用 ATmega32U4 微控制器。ATmega32U4 类似于 ATmega328，但前者集成了一个完整的 USB 转串行接口组件，并淘汰了 Uno、Duemilanove 等开发板上常见的一部分集成电路（IC）。

此外，采用 ATmega32U4 微控制器的开发板在编程接口的表现方式上略有不同。但对大多数人而言，这些不同在很大程度上是透明的。第 2 章将讲解 AVR 微控制器的一般功能，第 3 章将介绍可以在 Arduino 设备中见到的特定 AVR MCU 类型，第 4 章将讲解几种最重要的 Arduino 电路板，包括其引脚定义。

1.3　Arduino实物展示

表 1-2~ 表 1-5 展示了几种常见的 Arduino 开发板，其中包括了新老版本，但并不包括所有 Arduino 类型，因为一些新型开发板以及现有开发板的更新版本会定期推出。图 1-1 展示了各种 Arduino 开发板在外形与应用程序上的多样性。

从外形来看，Arduino 并不很大，其基线板大小约为 2.1 in × 2.7 in（53.3 mm × 68.6 mm），一般配有物理引脚，用于支持扩展板（参见第 8 章）。图 1-1 展示了两种 Arduino 开发板，旁边有直尺便于比较大小。图 1-2 展示了一个安装在免焊面包板上的 Nano 开发板。

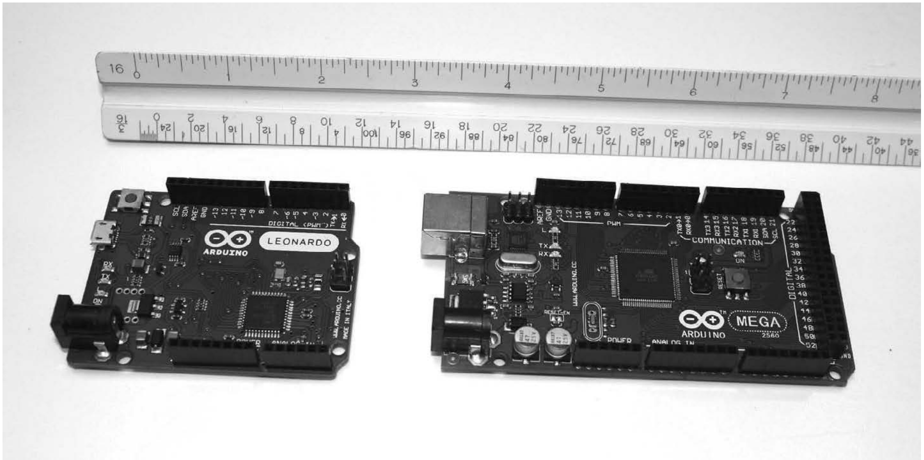


图 1-1：Arduino 开发板的相对大小

第 4 章包含大部分常见 Arduino 开发板的大小与引脚定义的参考图纸。请注意，有些 Arduino 开发板（比如 Nano）尽管很小，但同样拥有类似于 Duemilanove 的所有能力，只是没有方便的针脚插座与普通的 USB 连接器（B 型）。在需要使用小开发板且安装后不希望再有其他干扰的场合下，使用小型开发板是一个理想的选择。我们能想到的应用场景有自主收集环境数据的设备（比如太阳能驱动的气象数据自主监测站、海洋数据收集浮标）、火箭模型计时与数据收集系统、安保系统，甚至一台智能咖啡壶。

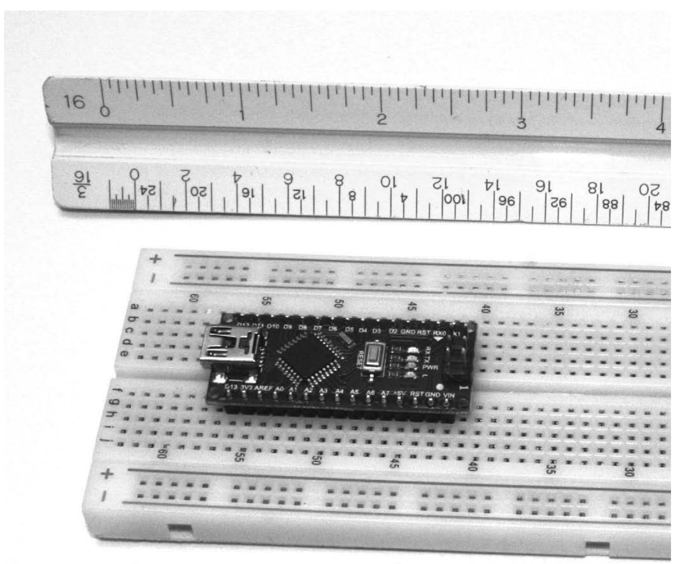


图 1-2：安装在免焊面包板上的 Arduino Nano

表1-2：Arudino开发板的基线布局

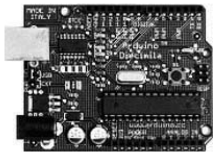
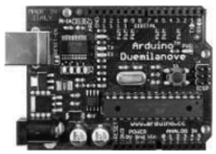
	型号	生产年份
	Diecimila	2007
	Duemilanove	2008
	Uno (R3 version)	2010
	Ethernet	2011
	Leonardo	2012

表1-3：Arduino Mega开发板的布局

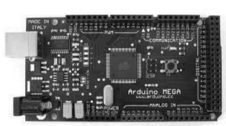
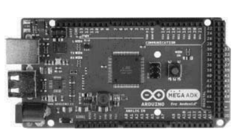
	型号	推出年份
	Mega	2009
	Mega 2560	2009
	Mega ADK	2011

表1-4：Arduino小型开发板

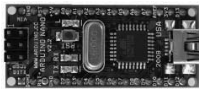
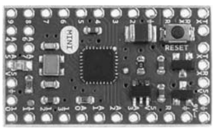
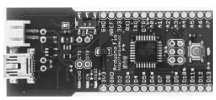
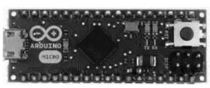
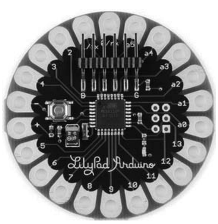
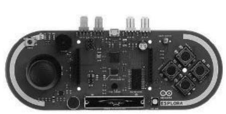
	型号	推出年份
	Nano	2008
	Mini	2008
	Fio	2010
	Mico	2012

表1-5：Arduino特别版

	型号	推出年份
	LilyPad	2007
	Esplora	2012

1.4 Arduino兼容设备

除了 Arduino.cc 设计或认可的各种开发板之外，还有许多设备与 Arduino 在硬件或软件上兼容。这些设备之所以能与 Arduino 兼容是因为，它们内部都集成了 Arduino Bootloader（或者类似的 Bootloader）。我们可以在 Arduino IDE 的下拉列表中选择合适且相互兼容的 Arudino 开发板，然后在 Arduino IDE 中为它们编写程序。

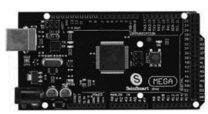
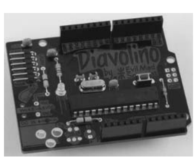
1.4.1 硬件兼容设备

对于 Arduino 硬件兼容设备，其上各种 I/O 引脚的排列布局方式与现有 Arduino 相同。一般而言，在硬件兼容开发板上，我们也可以使用为官方 Arduino 开发板制作的各种扩展板

与插件模块，背后原因将在 1.5 节讲解。

大多数情形下，硬件兼容开发板看上去与 Arduino 开发板非常类似，但其上没有官方 Arduino Logo 与丝印图形。而其他一些硬件兼容产品则看上去与标准的 Arduino 开发板完全不同，但它们采用了相同的排列布局方式提供引脚插口，这样用户就可以在其上使用标准的 Arduino 扩展板。还有一些硬件兼容产品配有额外的连接器，比如 SainSmart Uno 就带有额外的连接器，用于支持 I/O 功能。表 1-6 列出了几种常见的 Arduino 克隆板与兼容板。此外还有许多与 Arduino 相兼容的开发板，它们都是类似的，表中列出的这些可以让你有大致的了解。

表1-6：Arduino硬件兼容设备

	名称	类型	产地
	SainSmart UNO	Uno clone	中国
	SainSmart Mega2560	Mega 2560 克隆板	中国
	Brasuino	类似 Uno，稍作改动	巴西
	Diavolino	Arduino 布局兼容克隆套件	美国

请注意，Diavolino 是一个套件，需要用户自己组装。

1.4.2 软件兼容设备

除了硬件兼容设备之外，还有许多 Arduino 软件兼容开发板可以使用。这些开发板使用 Arduino 的 Bootloader 与开发环境，但实体大小与 Arduino 完全不同。软件兼容设备可以借助 Arduino 开发工具进行编程，但可能采用不同的 I/O 引脚排列布局，或者使用一些其他类型的连接器，用以取代旧 Arduino 开发板的引脚插口。对于基于 AVR 微控制器并嵌入一些较大设备或系统的定制电路板，如果微控制器中安装有 Arduino Bootloader，我们也会将其归入软件兼容设备之列。

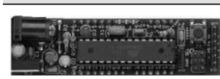
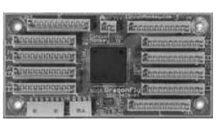
Arduino 的核心是处理器与预安装的 Bootloader。从这一定义上看，最简单的 Arduino 只有一个最基本的 ATmega AVR IC，其中装有 Arduino 固件，并且能够与免焊面包板和 Arduino 开发环境一起使用。你可以从多种渠道购买带有预载 Bootloader 代码的 AVR MCU IC，或者也可以自己动手制作。第 5 章将讲解装载 AVR MCU（带有 Arduino Bootloader 固件）的必需步骤。

有意思的是，从是否使用标准 I/O 连接器布局来看，一些 Arduino 官方开发板也不是硬件兼容的，比如 Mini、Micro、Nano、LilyPad、Esplora。它们不能直接与常见的扩展板一起使用，但仍然是 Arduino 开发板，并且都被 Arduino IDE 支持。

Adafruit Industries 推出的 Boarduino 就是一种 Arduino 软件兼容设备，它被设计安装在标准的免焊面包板上，很像一块全尺寸的 40-pinIC。Boarduino 有两种版本可用，DC 版本与 USB 版本。其中，DC 版本不带有板载 USB 芯片，为它编程时需要使用外部 USB 适配器。另一个软件兼容开发板的例子是 Circuit Monkey 出品的 Dragonfly 开发板，它使用标准的 Molex 连接器，代替 Arduino 常用的引脚与插口。Dragonfly 主要用于高振动环境，比如无人机（UAV）与机器人。

Raspduino 被设计用来安装在 Raspberry Pi 板上，在功能上等同于一个 Arduino Leonardo。这样形成的组合体大致等同于 Arduino Yún，但并不完全相同。每种设置各有优劣，表 1-7 列出了几种常见的 Arduino 软件兼容开发板。

表1-7：Arduino软件兼容设备

	名称	描述	产地
	Boarduino DC	被设计用于免焊面包板	美国
	Boarduino USB	被设计用于免焊面包板	美国
	Dragonfly	使用 Molex I/O 连接器	美国
	Raspduino	被设计用于 Raspberry Pi 板	荷兰

上面这些只是各种可用软件兼容开发板的一小部分。由于 AVR 微控制器很容易被集成到电子设计中，因此它广泛见于大量应用。借助 Arduino Bootloader 固件，对设备进行编程变得非常简单，设计也充满了无限可能。

1.5 Arduino命名约定

当 Arduino 电路设计与软件被开源时，Arduino 团队保留了对 Arduino 一词的使用权，将

其用于自己设计的产品，并为 Arduino Logo 注册了商标。正因如此，有时你会看到一些在行为和外观上与 Arduino 官方设备类似的产品，但其实它们并不属于 Arduino 品牌，也不是 Arduino 团队制作的。有些产品使用 -duino 或 -ino 作为名称的一部分，比如 Freeduino、Funduino、Diavolino、Youduino 等，而有些（像 SainSmart 制作的开发板）则只使用型号名称（如 Uno、Mega2560）。

> 写作本书时，Arduino LLC（由创始人创立的公司）与 Arduino SRL（由创始人之一创建的另一个公司）两家公司在 Arduino 商标使用的问题上存在争议。最终，Arduino LLC 获得了 Arduino 在美国的使用权，并获得 Genuino 在其他地区的使用权。

有时会有人生产一个开发板并声称是 Arduino，但实际上它只是一个未经许可而非法使用 Arduino 商标的克隆板。印制 Logo 所用的丝印层以及出现在官方 Arduino 上的其他信息也受版权保护，并且 Arduino 也没有发布带有 PCB 布局文件的丝印层。Massimo Banzi 在他的博客（http://www.massimobanzi.com）中专门针对这些未授权的开发板开设了一个版块，毫不夸张地说，他对无耻复制的调查是很有意思的。要查看相关内容，只要搜索 hall of shame 标签即可。

底线是，你可以复制 Arduino 的电路图、Bootloader 代码、Arduino IDE，并使用它们创建你自己的 Arduino——毕竟 Arduino 是开源的。但请不要把它叫作 Arduino，也不要未经 Arduino.cc 的许可而使用底片（artwork）。

1.6 使用Arduino可以实现的目的

除了容易编程之外（借助于 Arduino IDE），Arduino 的另一大特征是其电源以及依赖的 AVR 微控制器的性能。借助于一些容易获得的扩展板（第 8 章）与各种廉价的传感器、驱动器模块（第 9 章），真的没有什么是使用 Arduino 无法做到的，但前提是你要记住 Arduino 的一些基本限制。

首先是内存。AVR MCU 没有很多内存用来存储程序与变量，并且许多 AVR 部件并不支持用户通过任何途径添加更多内存。尽管如此，ATmega32 与 ATmega128 依然允许用户使用外部内存，但引脚的 I/O 功能将不再可用。Arduino 开发板的设计使得它不能使用外部内存，因为一个基本的设计假设是：AVR 芯片自身拥有必需的 I/O，并且用户运行的程序也相对较短。Arduino 并不能取代一个完整的计算机系统，这些计算机系统往往带有数千兆内存与硬盘驱动器（HDD）。市面上也有廉价的、基于 Intel 的单板计算机符合上面的描述，但它们并不适合放入一只旧的薄荷铁盒中，也不适合放入绑在一根柱子或树上的 PVC 管中，更不适合用于小型机器人或者作为火箭模型的负载部分，而 Arduino 可以。

第二个限制是速度。Arduino CPU 时钟频率通常介于 8 MHz ~20 MHz（第 4 章中有对各种 Arduino AVR 设备的详细比较）。这看上去很慢，但你必须认清两个关键事实：首先，AVR 采用了非常高效的 RISC（精简指令集计算机）设计；其次，从微控制器的视角看，现实世界中事情发生的频率通常不会太快。比如，所谓的智能调温器要多久才对家中或办公室中的温度进行一次采样呢？ 1 s 采集一次可能有些过分，每 5 s 或 10 s 采集一次的话，

系统会工作得很好。机器人需要多久发出一次超声波脉冲以检测前方是否有障碍物呢？每 100 ms 发送一次脉冲可能就足够了（除非机器人移动得非常快）。对于一台以 16 MHz 运行的 Arduino（例如 Arduino Leonardo），两次传感器脉冲之间大约会有 100 万次以上的 CPU 时钟节拍，具体要取决于 CPU 与脉冲有关的工作内容。考虑到 AVR 能够在一个或两个时钟周期内执行许多指令，这意味着超声波传感器的每个脉冲之间有大量可用的 CPU 活动。

第三个限制是电力供应。由于 Arduino 硬件实际上只是一个装有 AVR IC 的 PCB，所以微控制器与外部世界之间不存在缓冲。通过 AVR 的电流过大会导致 AVR 烧毁，也就是说，要避免 IC 因过热而损坏，你必须多留心，不要让流经设备的电流超过设备本身的承受能力。还需要考虑电压，有些 AVR 拥有 3.3 V 的 I/O，而其他一些则可以承受 5 V 电压。把 5 V 的晶体管 - 晶体管逻辑电路（TTL）连接到 3.3 V 设备上通常会导致硬件故障，甚至损坏。

请牢记上面这些限制，下面只列出了 Arduino 几种可能的应用场景。

- 真实世界监控
  - 自动天气监测站
  - 闪电探测器
  - 太阳能电池板太阳追踪系统
  - 背景辐射监测
  - 野生动物自动检测仪
  - 家用或商用安全系统
- 小型控制
  - 小型机器人
  - 火箭模型
  - 飞机模型
  - 四旋翼无人飞行器
  - 制造小型机床的简易 CNC
- 小型自动化控制
  - 自动化温室
  - 自动化水族箱
  - 实验室梭式机器人
  - 精密温控箱
  - 自动电子测试系统
- 艺术表演
  - 动态灯光控制
  - 动态声音控制
  - 运动结构
  - 观众互动作品

第 11 章 ~ 第 13 章将介绍 Arduino 的具体应用示例，比如智能调温器、可编程信号发生器、自动火箭发射控制系统，以满足你对小轨道飞行的渴望。这些应用只是冰山一角，Arduino 应用有无限可能，事实上它只受限于你的想象力。只要你不把 Arduino 当作完整的计算机系统来用，就可以把它集成到各种有趣的应用场景中，就像 Arduino.cc 希望你所做的一样。

1.7 更多信息

本章列出的开发板只是众多开发板中的一小部分，关于 Arduino 还有更多故事。在谷歌搜索框中输入 Arduino 关键字，你将看到成千上万的参考页面，从中能学到更多有关 Arduino 的内容。

Arduino 官方站点为：http://www.arduino.cc

Massimo Banzi 博客地址为：http://www.massimobanzi.com

此外，查看附录将看到更多相关网站链接与参考书目。

第 2 章

AVR微控制器

因为基于 AVR 的 Arduino 其实只是一个 AVR 微控制器的物理平台（例如分接板），所以 Arduino 的电气特性本质上就是 PCB 上 AVR 设备的电气特性。理解 Arduino 的底层细节其实就是理解 AVR 设备的问题。因此，本章是对 AVR 系列使用的主要功能的高级说明，包括 AVR CPU、所谓的“外围设备”功能，比如定时器、计数器、串行接口逻辑、模数转换器（A/D）、模拟比较器、离散数字 I/O 端口。

AVR 微控制器广泛适用于各种配置与封装类型，这让写作本章成为一个挑战。幸运的是，各种类型的 8 位 AVR 设备使用通用的 CPU 以及模块化的内部架构，各个组成部件都围绕着内部数据总线进行搭建。这种模块化的架构方式允许设计者把不同的组合包含到设计中，并把大量功能模块加入到 AVR 的内部电路，以迎合特定的设计需求，生产出能够满足不同应用场景的产品。

由于篇幅所限，本章只做简要讲解，并把重点放在基本特性上，不会过多涉及底层细节。关于底层细节的更多内容，各位可以在 Atmel（http://www.atmel.com）的参考文档中找到。如果想了解更多有关 AVR 微控制器的逻辑电路与寄存器级别的细节信息，请参考 Atmel 的数据表、用户手册、应用指南，它们都是免费的。

2.1 背景

AVR 微控制器诞生于 20 世纪 90 年代早期，起初只是挪威理工学院的一个学生项目。Alf-Egil Bogen 和 VegardWollan 在挪威特隆赫姆市的一家半导体设备公司工作期间，创造了一个带有 RISC 内部架构的 8 位设备。之后，他们把它卖给了 Atmel 公司，Bogen 与 Wollan 继续努力进行改善。

AVR 微控制器是高度可配置的，并且具有非常好的通用性，它们拥有几种独特的特性，使其不同于其他 8 位微控制器，比如 8051 或 68HC05 组件。AVR 是一种经过改良的哈佛架构的 8 位 RISC 微控制器。在哈佛架构的只读程序中，代码与可修改数据（变量）分别存储在不同的内存空间中。相比之下，68040 这类微处理器使用冯·诺依曼体系架构，程序与数据共用相同的内存空间。

AVR 系列设备也是首个使用板载 flash 内存存储程序的设备之一，它摒弃了其他微控制器中常见的一次性可编程 ROM（只读内存）、EPROM（可擦可编程序只读存储器）或 EEPROM（带电可擦写可编程只读存储器）。这使得对 AVR 微控制器进行重新编程变得简单，只需将新的程序代码装入设备内部闪存即可。大部分 AVR 部件都带有少量 EEPROM，用于存储操作参数等，它们在闪存发生改变时必须一直存在。

2.2　内部架构

从内部结构看，AVR ATmega 微控制器由 AVR CPU、各种输入 / 输出、时序、模数转换、计数器 / 定时器、串口功能以及其他各种部件功能组成，Atmel 将其称为“外围功能”（peripheral function）。除了 I/O 功能之外，不同 AVR 微控制器的主要差别在于板载闪存（flash memory）与 I/O 功能的数量。所有 8 位部件基本都使用相同的 AVR CPU 核心。下面给出了 AVR 微控制器的一些基本特性。

- RISC 架构
  - 131 条指令
  - 32 个 8 位通用寄存器
  - 最大时钟频率为 20 MHz（20 个 MIPS 操作）
- 板载内存
  - 快速程序存储器（多达 256 kB）
  - 板载 EEPROM（多达 4 kB）
  - 内部 SRAM（多达 32 kB）
- 工作电压
  - VCC=1.8 V~5.5 V DC

图 2-1 是 AVR CPU 核心的简单框图，AVR CPU 核心常用在 8 位 AVR 设备中。虽然图 2-2 是 AVR 设备的通用高级框图，但请注意，它并不代表任何特定 AVR 设备，描述的只是通用的 AVR 设备。

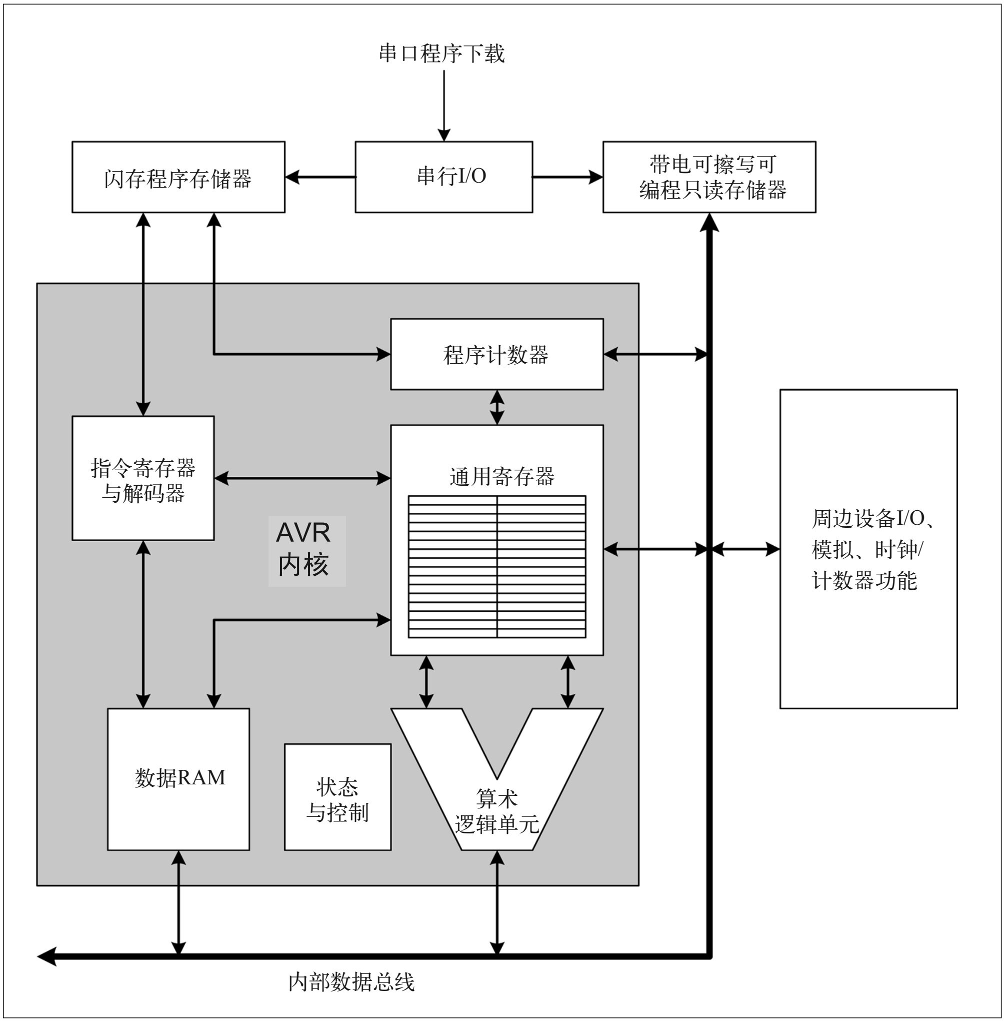


图 2-1：AVR CPU 框图

外围功能由 CPU 通过内部高速数据总线进行控制。控制寄存器（独立于 CPU 寄存器）用于配置外围设置的操作。所有外围共享端口引脚，并分别带有独立的数字 I/O 功能。

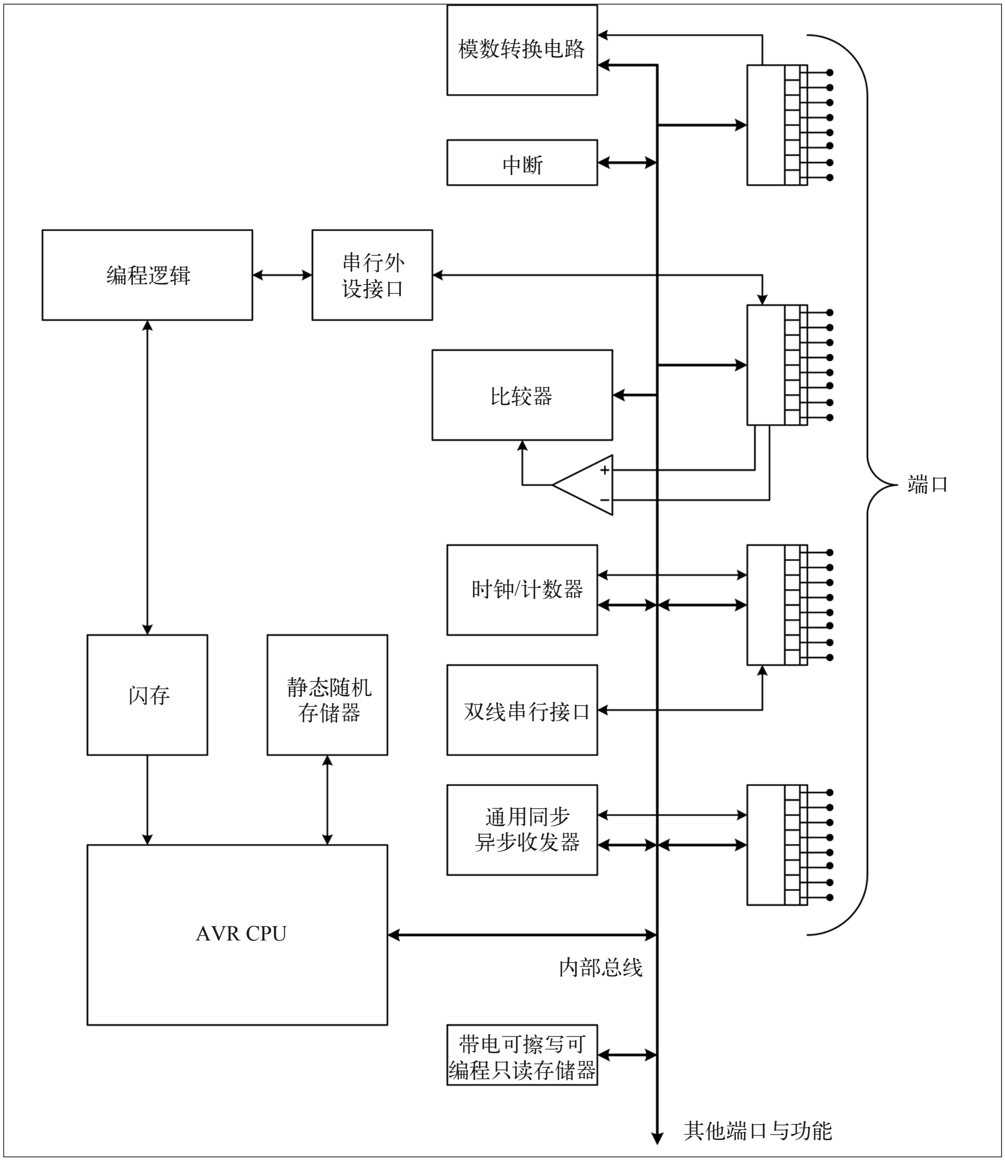


图 2-2：通用 AVR 微控制器框图

Atmel 制造了许多不同类型的 AVR 微控制器，硬件设计师可以灵活选用，以便更好地满足自己的特定需求，并减少未用引脚的数量以及印刷电路板上的空间浪费。一些器件——比如 tinyAVR——采用小型表面贴装封装，只带有 6 个引脚。每个引脚都有一个或多个独立的数字 I/O 端口，通过编程可以执行多种功能（参见 2.4 节）。

例如，ATTINY13-20SQ 采用 8 引脚 DIP（双列直插式封装）或 SOIC（小外形集成电路）表面贴装封装。设备的 6 个引脚连接到一个 8 位的内部 I/O 端口（端口 B），其余两个引

脚是 VCC（电源）与地。6 个端口 B 引脚能够被配置为模拟输入、振荡器输出、中断输入、SPI 信号或独立的数字输入输出。尽管 ATTINY13-20SQ 很小，但从内部结构看，它仍然是 AVR 微控制器，并带有用于存储程序的 1 kB 大小的内置闪存，以及用于存储变量的 64 B 的 RAM。

另一方面，还有一些 AVR 器件，比如 ATmega649，带有 9 个 8 位的端口、64 kB 大小的闪存、4 kB 大小的 RAM、2 kB 大小的 EEPROM、54 个通用 I/O 引脚，以及一个集成的 LCD 接口。AVR32 系列采用 32 位的 AVR 处理器，拥有多达 256 kB 的闪存、32 kB 的 RAM、一个集成的 DSP 单元（处理数字信号）、受保护内存、36 个通用 I/O 引脚。

虽然 Arduino 开发板并未使用类似 tinyAVR（将 Arduino bootloader 放入 1 kB 大小的闪存并为有用程序留下空间的确是一个很大的挑战）的小型 AVR 器件，也未采用 ATmega649、AVR32，但我真正想说的是，AVR 系列为用户提供了多种选择，Arduino 设备选用的器件并不是唯一可用的 AVR 器件。

2.3　内部存储器

所有 AVR 设备都包含不同数量的存储器，这些存储器大致可以分为 3 种类型：闪存（flash）、SRAM（静态随机存取存储器）和 EEPROM。其中，闪存存储程序代码，SRAM 存储临时数据（如程序变量、栈），EEPROM 存储软件变更与开关机过程中需要持续保留的数据。闪存与 EEPROM 能从外部加载数据，并且当 AVR 断电时，它们仍然能够保留其中的数据。而 SRAM 是易挥发的，当 AVR 断电时，其中保存的内容就会丢失。

2.4　外围功能

AVR 微控制器的"心脏"是一块 8 位的 CPU，但让它真正有用的是其内置的外围功能，这些功能被集成到 IC，并带有 CPU 逻辑。不同类型的 AVR 设备，其外围功能也有所不同。有些只有一个定时器，而有些则有两个或更多（有的多达 6 个）。有些拥有 10 位的 A/D 转换器（ADC），而另一些的则为 12 位。所有 AVR 器件都为离散的数字信号提供双向 I/O 引脚。有些版本也提供触屏支持，以及其他一些交互接口。

本节只介绍通用的外围功能，它们被用在 Arduino 产品使用的各种 AVR 设备中。本节以 ATmega168 为例子进行讲解，不会提供每种 AVR 微控制器的详细参考，但会讲解每种外围设备的一般功能。相关内容请参考第 3 章（介绍 Arduino 开发板所用处理器的细节信息），也可以查阅 Atmel 技术文档，以了解更多本书未提及的底层技术细节。

2.4.1　控制寄存器

除了 CPU 中的 32 个通用寄存器之外，AVR 设备也含有一些控制寄存器，它们用于控制 I/O 端口、定时器、通信接口以及其他功能的工作方式。控制寄存器的设置因设备类型的不同而不同，因为不同类型的设备可能会有不同数量的端口，以及不同的外围功能配置。本书介绍的 AVR 器件（指 Arduino 开发板上使用的 AVR）的控制寄存器信息参见附录 B，也可以直接在 Atmel 的文档中找到相关的技术细节描述。

即使是 ATmega168 这类略显保守的 AVR 器件，也拥有很多内部功能。若每个引脚只分配一项功能，那么它所拥有的引脚是远远不够的。由于这个原因，AVR 微控制器上的大多数引脚都能被重新配置为“基于控制寄存器中的设置执行特定功能”。因为引脚功能是可动态配置的，所以我们可以让一个引脚在某个时间点执行一种功能，然后当控制寄存器的值发生改变时，使其执行另一项功能。

比如，ATmega168 采用的是 28 引脚的 DIP 封装方式，它的第 12 号引脚连接到 PD6（端口 D，位 6），但它也可以被配置为中断源（PCINT22）、AVR 内部模拟比较器（AIN0）的正向输入，或者用作定时器比较逻辑电路（Timer/Counter0 输出比较匹配 A）的输出，用于产生 PWM（脉冲宽度调制）信号。

2.4.2 数字I/O端口

AVR 微控制器使用双向 I/O 端口与外部世界进行通信。端口是一个 8 位的寄存器，它的部分位或所有位会被连接到 AVR 设备封装的物理引脚上。不同类型的 AVR 设备拥有不同数量的端口，ATTINY13-20SQ 只有 1 个，ATmega649 多达 9 个。每个端口都使用 A、B、C 等标签进行标示。

端口的每个引脚都由内部逻辑进行控制，这些内部逻辑管理着信号方向、内部上拉电阻的状态、计时以及其他功能。图 2-3 是 AVR I/O 端口的简图，图中 Px 表示 port bit/pin x（0~7）。关于 AVR 端口逻辑的更多细节描述，请参考 AVR 技术文档。

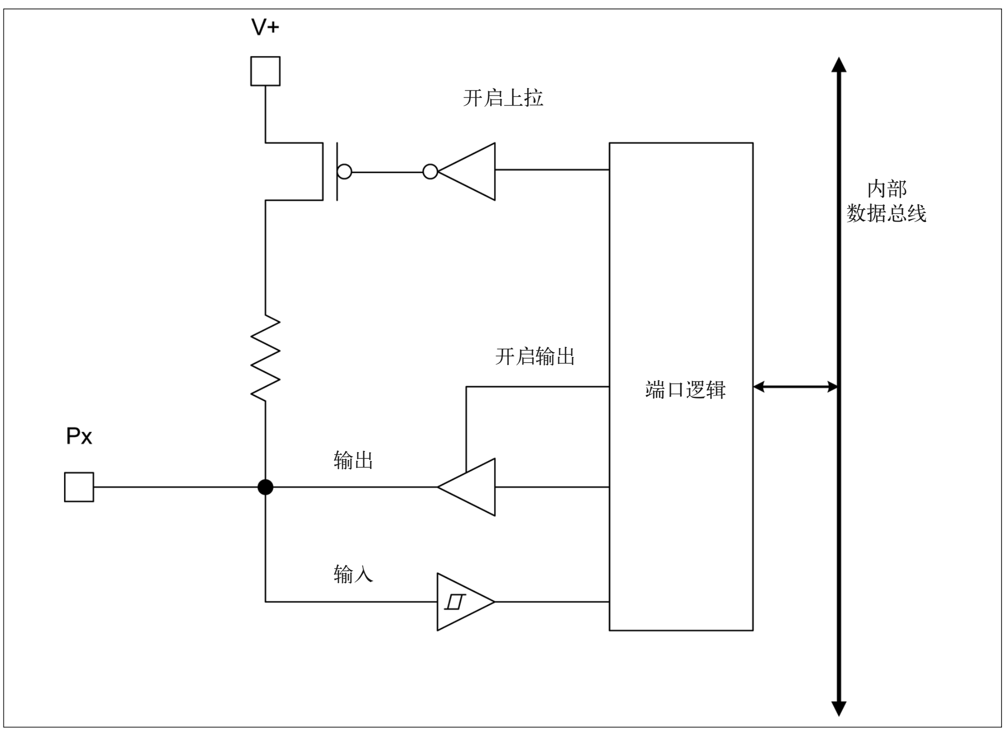


图 2-3：AVR I/O 端口框图

由于功能控制逻辑复杂且巧妙，所以 AVR 端口能够执行多种不同功能，其中一些功能可以同时进行。当对一个端口进行配置并将其用作输出时，我们仍然可以借助它读取数据，输出可以用来触发中断（2.8 节）。

2.4.3　8位定时器/计数器

AVR 微控制器中，有两种形式的 8 位定时器 / 计数器可用。第一种类型中，时钟输入来自于主系统时钟，因此定时器 / 计数器是同步的；第二种类型中，可以使用外部时钟源在异步模式下进行操作。图 2-4 是 AVR 定时器的简图。你可以在附录 B 找到定时器 / 计数器的控制寄存器的定义，在 Atmel 技术文档中可以看到细节描述。

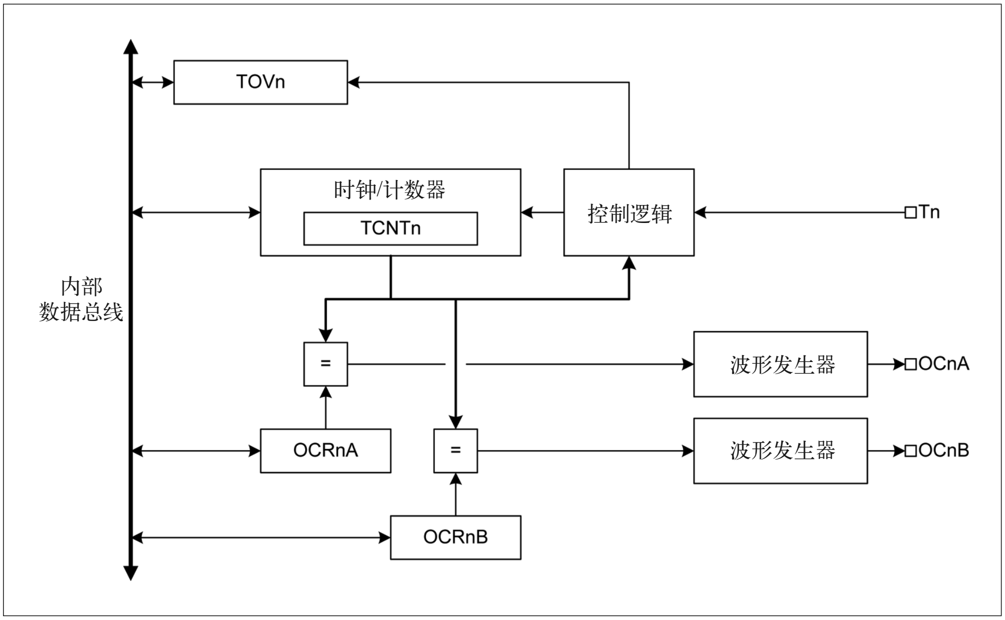


图 2-4：AVR 定时器 / 计数器框图

在 AVR 8 位定时器 / 计数器周边电路（peripheral function）中，Timer/Counter0 模块是一个通用的定时器 / 计数器。其特色是带有两个独立的输出比较电路，拥有 4 种操作模式。

正常模式（**Normal mode**）

这是最简单的定时器 / 计数器操作模式。计数总是在增长，当计数器达到最大 8 位值时，不会对计数器执行清空操作。在此情形下，计数器会发生溢出并归 0。计数器归 0 时，就会设置 Timer/Counter 的溢出标志位（TOV0）。TOV0 标记对应于第 9 位，它只在定时器溢出时被设置，而不是被清除。定时器溢出中断会自动清空溢出标志位，中断也可以用来增加内存中另一个基于软件的计数器。在任何时候，新的计数器值都可以被写入 TCNT0 寄存器。

CTC（Clear Timer on Compare）模式

在 CTC 模式下，OCR0A 寄存器通过定义计数器的最大值操纵计数器的分辨率。这有利于更好地控制比较匹配输出频率，也有助于简化外部事件计数。

快速 PWM 模式

快速脉冲宽度调制模式支持产生高频 PWM 波形。

相位校正 PWM 模式

相位校正 PWM 模式提供一个产生高分辨率相位校正 PWM 波形的选项。

此外，一些 AVR 设备包含 8 位定时器 / 计数器，拥有通过外部时钟输入（TOSC1 与 TOSC2 时钟输入引脚）进行异步操作的能力。在功能上，它们等同于前面介绍的 8 位同步定时器 / 计数器电路。

2.4.4 16位定时器/计数器

16 位定时器 / 计数器与 8 位类似，但拥有更大的计数范围。它是真正的 16 位逻辑，允许产生 16 位可变周期的 PWM。该模块也具有两个独立的输出比较电路、双缓冲输出比较寄存器，以及一个抗噪的输入捕获电路。除了产生 PWM 之外，16 位定时器 / 计数器也能捕获高分辨率外部事件，产生频率以及信号定时测量，并且 16 位定时器 / 计数器也有能力产生 4 种不同的中断（TOV1、OCF1A、OCF1B、ICF1）。

2.4.5 定时器/计数器预分频器

在一个 AVR 设备中，一个或多个计数器可以共用相同的预分频器逻辑，但它们各自拥有不同的设置。本质上，预分频器是一个分频电路，它在 f/8、f/64、f/256 或 f/1024（被称为 tap）产生系统 I/O 时钟的“衍生物”。一个定时器 / 计数器可能使用 f/64 tap，而另一个可能使用 f/1024 tap。借助预分频器可以对定时器 / 计数器范围进行扩展，使之更接近外部事件发生的频率；也可以在定时器 / 计数器溢出与重置之间增加时间。

2.5 模拟比较器

AVR 微控制器的模拟比较器用于比较 AIN0 与 AIN1 引脚的输入电压。尽管 AIN0 被定义为正向输入，AIN1 被定义为负输入，但这仅仅用于反映它们之间的关系，并非指输入电压的实际极性。图 2-5 是 AVR 模拟比较器电路的简图。

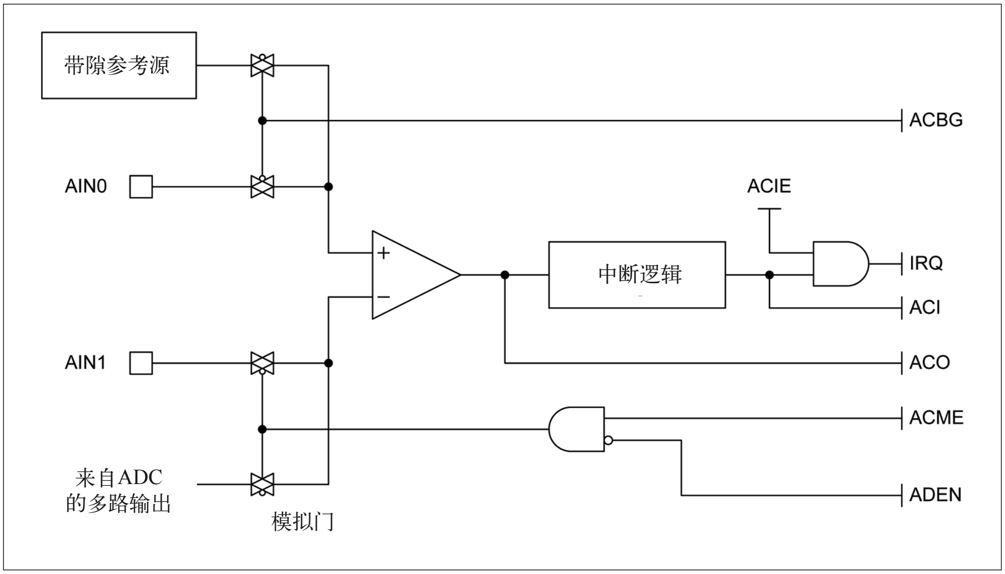


图 2-5：AVR 模拟比较器框图

当 AIN0 大于 AIN1 时，比较器逻辑将比较器标识设置为 ACO。我们可以对比较器的输出进行配置，使之触发一个定时器 / 计数器模块的输入捕获功能，也可以让它产生针对比较器的特定中断。中断事件也可以配置为“当比较器输出发生涨落或切换时进行触发”。

除了比较 AIN0 与 AIN1 输入的电压之外，模拟比较器电路还能做更多事情。我们也可以对模拟比较器的输入进行配置，使得可以将 AIN1 输入看作内部带隙基准电压，或者将 AIN0 看作 ADC 多路复用器的输出（并且该电压对于 ADC 输入仍然有效）。带有 4 个箭头的特殊符号是模拟门。模拟门对控制输入做出何种反应由反向循环（inversion circle）指示，即使用反向控制输入时，若控制为低电平，则传递模拟信号，否则传递数字信号。

2.6 模数转换器

大多数 AVR 微型控制器包含 8 位、10 位或 12 位模数转换器。在 ATtiny6 与 ATtiny10 器件中可以看到 8 位转换器，而一些 AVR 微控制器的汽车专用版本则不包含 ADC。

当 ADC 是 AVR 设计的一部分时，它会有 4~28 个输入，实际可用的输入数量在很大程度上取决于所用的物理封装。通过内部多路复用器，每次只选一个输入。也就是说，它们不能同时处于激活状态。此外，ADC 输入多路复用器所使用的一些 I/O 引脚也可以被指派为其他功能。

ATmega168 设备拥有 6 个或 8 个 ADC 输入通道，具体数量取决于所用的封装类型。PDIP 封装（塑料双列直插式封装）带有 10 位 ADC，它拥有 6 个输入通道。TQFP 与 QFN/MFL 表面贴装有 10 位 ADC，拥有 8 个输入通道。图 2-6 是 AVR ADC 外围电路的框图。

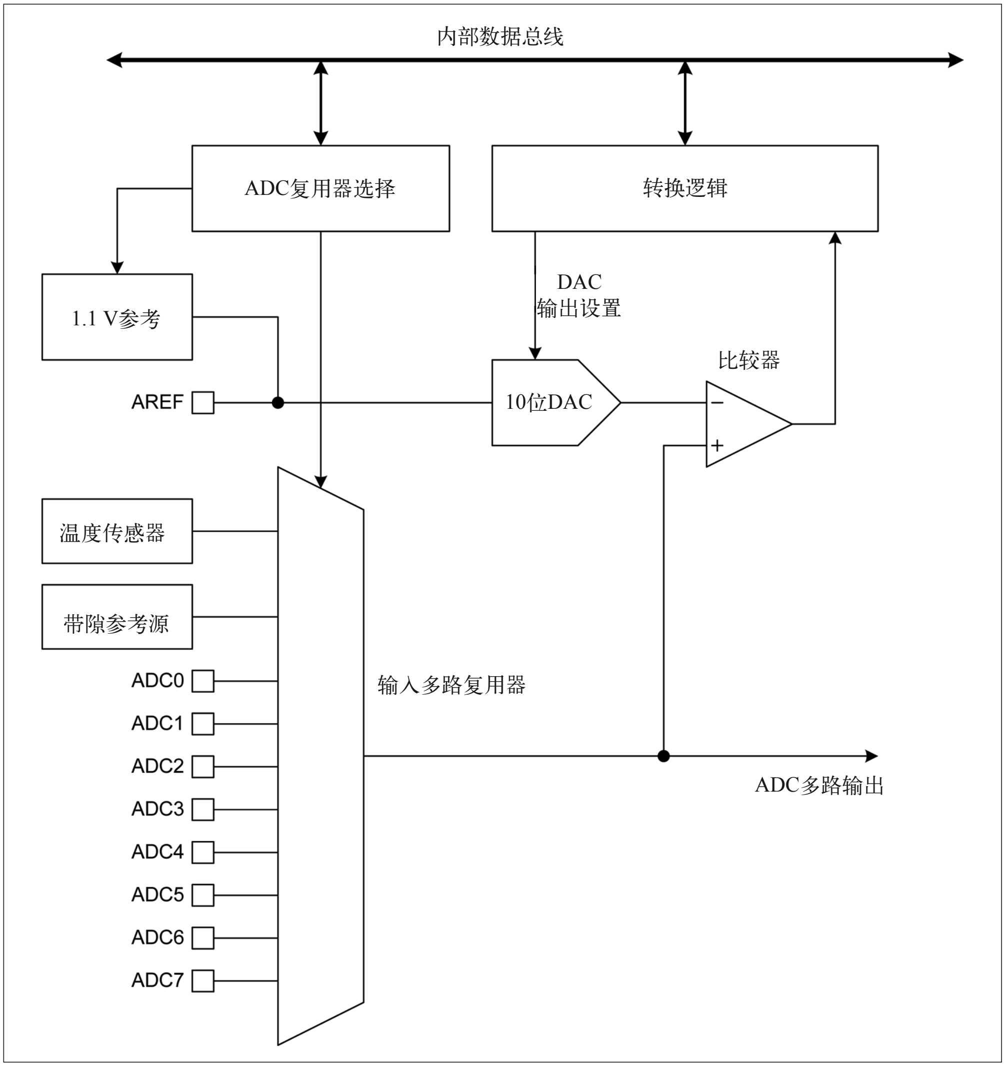


图 2-6：AVR 模数转换器框图

请注意，图 2-6 中，AVR 使用了名为“逐次逼近”（successive approximation）的转换器。这种类型的转换器速度不是特别快，但容易实现，只需要一个 DAC（数模转换器）与一个比较器。在自由运行模式、维持全分辨率的水平下，一个 10 位 AVR ADC 每次采样的标准转换时间大约为 65 μs。

2.7 串行I/O

ATmega168 主要提供 3 种串口：同步 / 异步串口、SPI 主从同步串口、面向字节的双线接口——类似于 Philips I2C（内部集成电路）标准。

2.7.1 USART

许多 AVR 器件都内置有一个通用组件，称为 USART（通用同步 / 异步收发器），亦称作 UART（通用异步收发器）。它能实现 RS-232 或 RS-485 接口，或者在没有芯片间通信（chip-to-chip communication）的外部接口逻辑时使用。波特率由微控制器使用的时钟频率决定，常见的波特率为 9600。并且，使用快速外部晶振能够获得更高的波特率。USART 也能用在 SPI（串行外围接口）模式下，或者 AVR 设备专用的 SPI 逻辑中。图 2-7 显示 AVR USART 外围电路的基本内部元件。

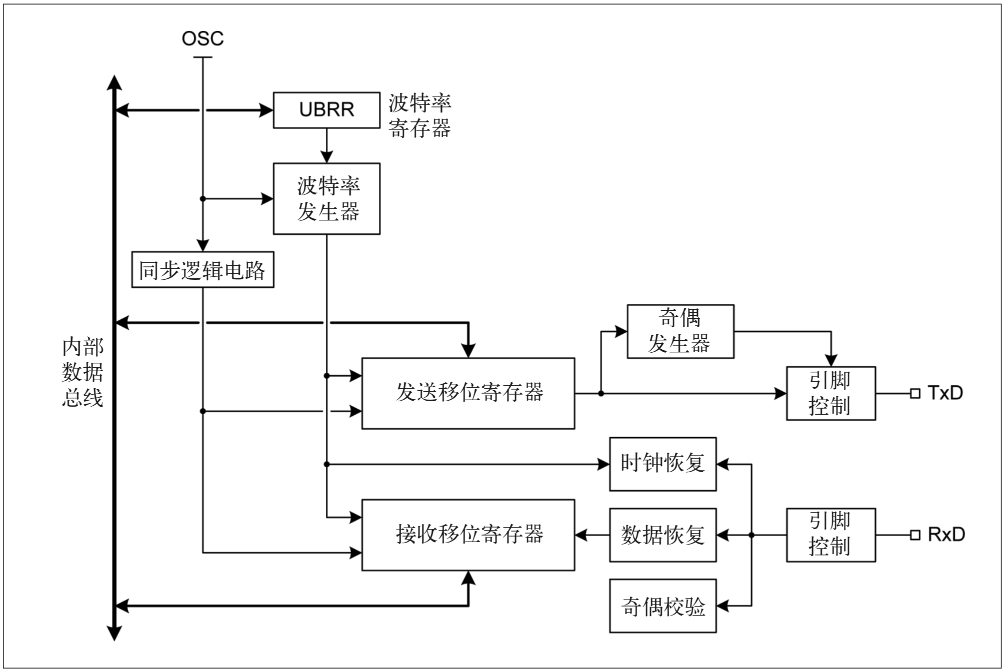


图 2-7：AVR USART 框图

2.7.2 SPI

AVR 的 SPI 外围逻辑支持所有 4 种标准的 SPI 操作模式。AVR 设备上的 I/O 引脚可以被重新配置，以便充当 SPI 使用的 MOSI、MISO、SCK[1] 信号。这些引脚不同于 USART 使用的 RxD 与 TxD 引脚（接收数据与传送数据）。图 2-8 显示了 SPI 逻辑的高级视图。

注 1：master out, slave in（串行数据输入信号线）、master in, slave out（串行数据输出信号线）serial clock（移位时钟信号线）。

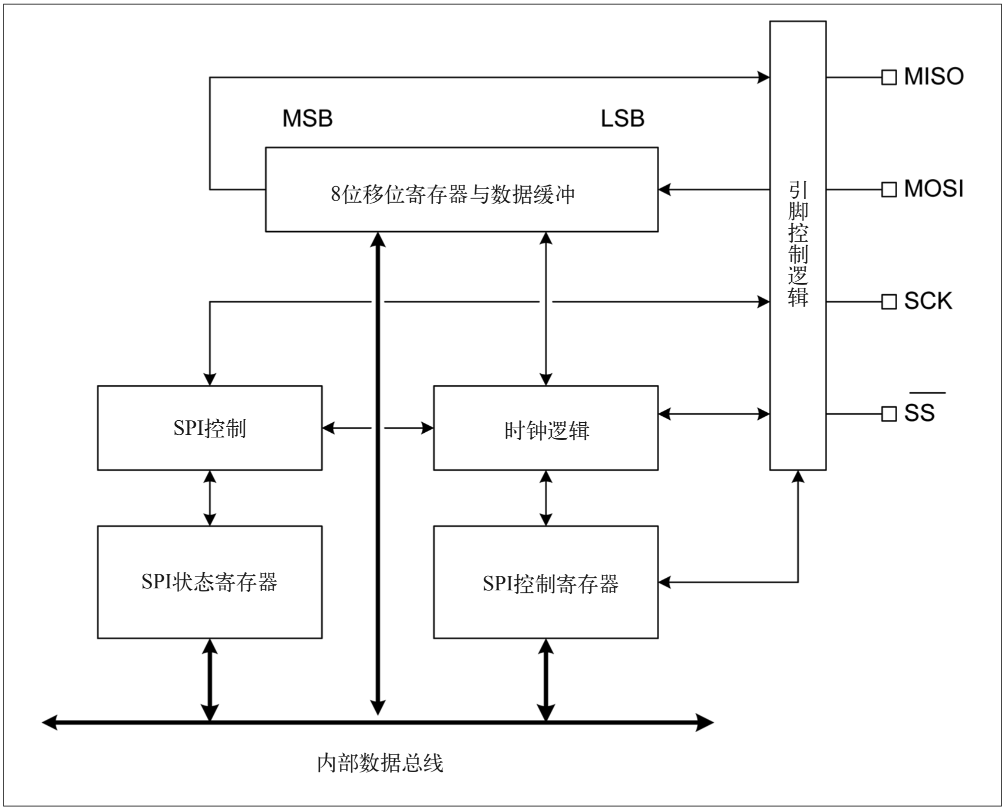


图 2-8：AVR SPI 框图

2.7.3 TWI

许多 AVR 设备还支持第三种串行 I/O，即双线接口（TWI）。该接口兼容 Philips I2C 协议，支持主 / 从操作模式以及 7 位设备地址。通过使用多主控总线仲裁（multimaster bus arbitration），TWI 接口的传送速度高达 400 kHz. 当 AVR 处于休眠模式时，它有能力产生唤醒条件。从内部结构看，TWI 外围设备相当复杂，与 USART 或 SPI 外围设备相比更是如此。图 2-9 显示的是 TWI 接口的结构框图。

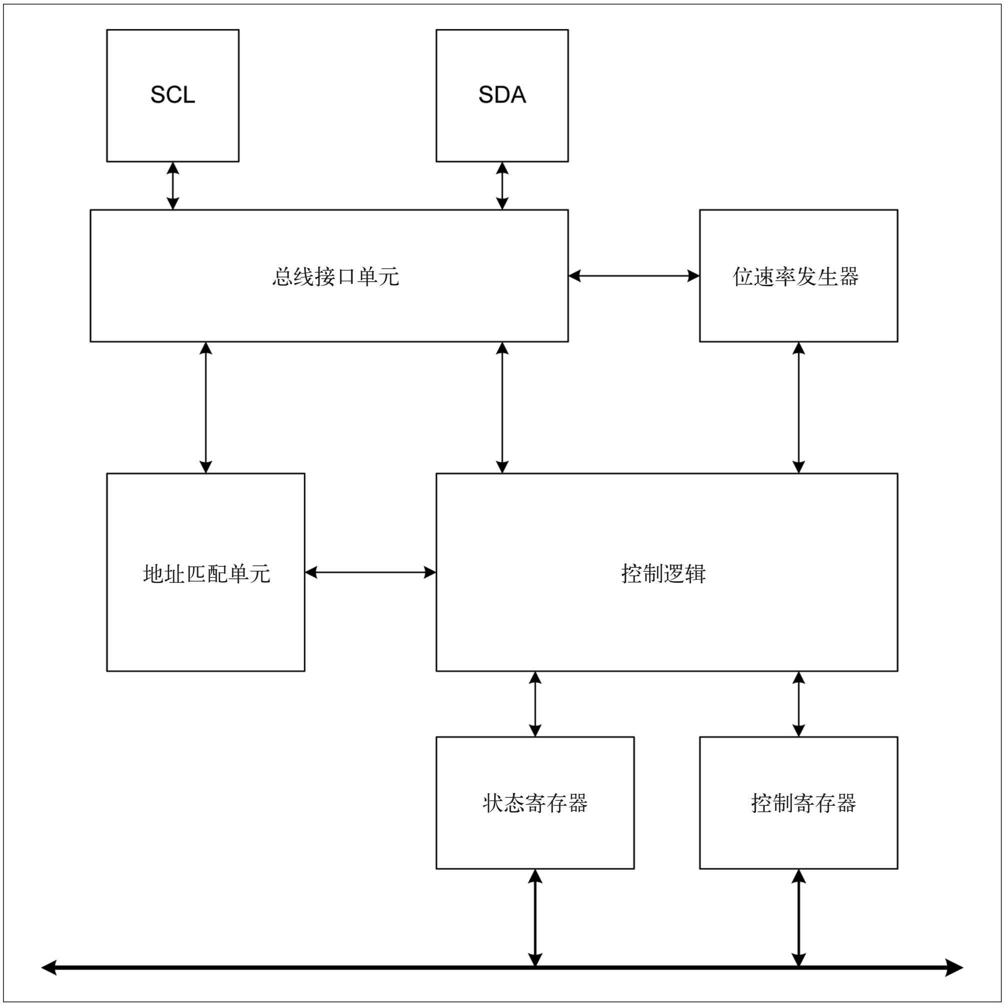


图 2-9：AVR TWI（I2C）框图

2.8 中断

中断是现代处理器的基本功能。它们允许处理器对内部或外部事件做出响应，通过切换到一段特殊的中断处理代码来处理中断。一旦中断处理代码执行完毕，控制流将返回程序中发生中断的位置，并继续向下执行。在 AVR 中，通过修改控制寄存器中相应的位，可以打开或关闭中断响应。下面将以 ATmega168 为例进行讲解。对于其他类型的微控制器，请参考附录 A 或者 Atmel 官方文档（http://www.atmel.com/design-support/documentation/）。

ATmega168 拥有两个外部中断输入：INT0 与 INT1。这些输入可以配置在下降沿、上升沿或低电平时触发。EICRA 控制寄存器（见附录 B）配置具体行为。INT0 与 INT1 需要存在 I/O 时钟。只要输入保持在低电平，那么低电平中断模式会产生中断。

ATmega168 I/O 引脚也能用作中断源。端口改变中断被定义为 PCINT0~PCINT23，每个都与设备的一个 I/O 端口引脚相关联。在启动状态下，每当端口引脚状态发生变化，就会产生一个中断，即便引脚配置为输出也是如此。在端口改变中断检测功能开启期间，当程序切换引脚状态时，这允许引脚在软件控制下产生中断。

当 PCINT0~PCINT7 的任意一个引脚发生改变，就会触发 PCI0 中断。而 PCINT8~PCINT14 的引脚则会触发 PCI1 中断，PCINT16~PCINT23 的引脚将触发 PCI2 中断。PCMSK0、PCMSK1、PCMSK2 寄存器控制着哪些引脚可以产生引脚改变中断。

当启用的中断发生时，CPU 就会跳转到内存向量表的一个位置上，该位置指派有特定中断。地址中包含着一个跳转指令（RJMP），指向处理该中断的实际代码段。当中断处理代码执行完毕后，执行流会返回原程序中发生中断的位置。图 2-10 中，可以看到中断向量表示如何把执行流切换到中断处理代码段，当中断处理代码执行完毕，再将控制流返回到主程序。

例如，ATmega168 的向量表拥有 26 条记录，如表 2-1 所示。对于其他类型的处理器，请阅读 Atmel 文档，获取有关中断的更多信息，并了解它们在 AVR 设备中是如何被管理的。

表2-1：ATmega 168中断向量

向量	地址	中断源	定义
1	0x0000	RESET	外部引脚、上电、电压降低
2	0x0002	INT0	外部中断请求 0
3	0x0004	INT1	外部中断请求 1
4	0x0006	PCINT0	引脚更改中断请求 0
5	0x0008	PCINT1	引脚更改中断请求 1
6	0x000A	PCINT2	引脚更改中断请求 2
7	0x000C	WDT	看门狗超时中断
8	0x000E	TIMER2	COMPA 定时器 / 计数器 2 比较匹配 A
9	0x0010	TIMER2	COMPB 定时器 / 计数器 2 比较匹配 B
10	0x0012	TIMER2	OVF 定时器 / 计数器 2 溢出
11	0x0014	TIMER1	CAPT 定时器 / 计数器 1 捕获事件
12	0x0016	TIMER1	COMPA 定时器 / 计数器 1 比较匹配 A
13	0x0018	TIMER1	COMPB 定时器 / 计数器 1 比较匹配 B
14	0x001A	TIMER1	OVF 定时器 / 计数器 1 溢出
15	0x001C	TIMER0	COMPA 定时器 / 计数器 0 比较匹配 A
16	0x001E	TIMER0	COMPB 定时器 / 计数器 0 比较匹配 B
17	0x0020	TIMER0	OVF 定时器 / 计数器 0 溢出
18	0x0022	SPI, STC	SPI 串行传送完成
19	0x0024	USART, RX	USART Rx 完成
20	0x0026	USART, UDRE	USART 数据寄存器空
21	0x0028	USART, TX	USART Tx 完成
22	0x002A	ADC	ADC 转换完成

（续）

向量	地址	中断源	定义
23	0x002C	EE READY	EEPROM 准备好
24	0x002E	ANALOG COMP	模拟比较器
25	0x0030	TWI	双线串口
26	0x0032	SPM READY	存储程序内存准备好

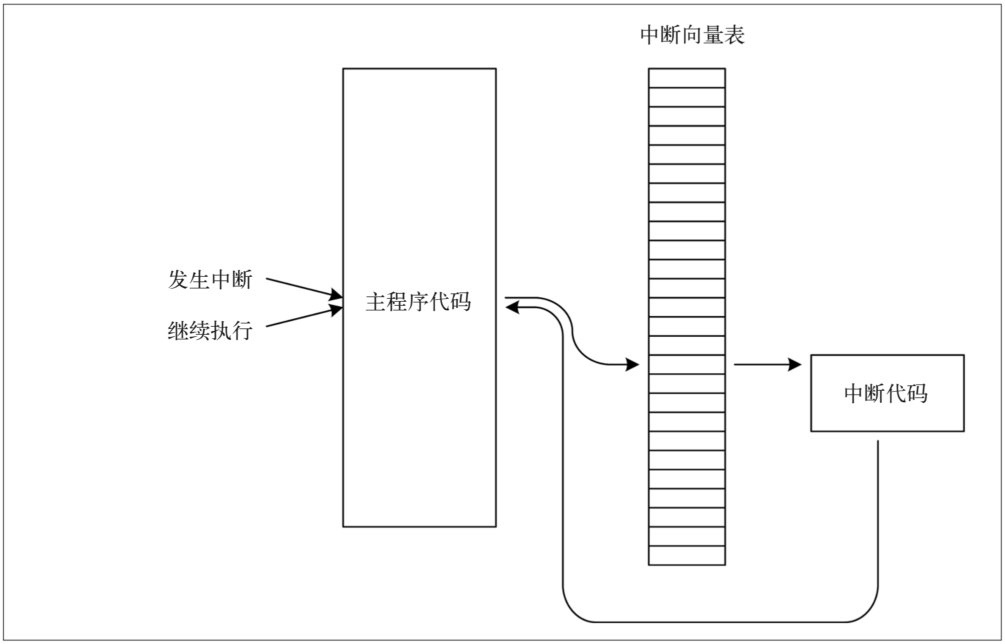


图 2-10：AVR 中断向量（ATmega168/328）

2.9 看门狗定时器

AVR 提供一个看门狗定时器（WDT），可配置的超时范围为 8 ms~16 ms。启用 WDT 后，用它产生超时重置、超时中断，或者两者的组合。WDT 使用独立的内嵌式振荡器（on-chip oscillator），并且由于采用独立的振荡器进行时钟控制，所以当微控制器处于睡眠状态时，它会一直进行计数。这样，我们就可以使用它在一段时间之后重新唤醒微控制器。

此外，当定时器超时，如果仍然没有来自软件的重置动作，也会常常使用 WDT 强制进行重置或产生中断。这有助于微控制器脱离死锁或检测失控的代码。

2.10 电气特性

ATmega168 与 ATmega328 AVR 微控制器能够在 1.8 V~5.5 V DC 的 VCC 之下工作。而 ATmega32U4 能够在 2.7 V~5.5 V DC 的 VCC 之下工作。

AVR 设备的电流消耗各有不同，这主要取决于设备类型、微控制器的激活或非激活（睡眠）状态，以及时钟频率。ATmega 与 XMEGA 系列产品的电流消耗范围为 0.55 mA~30 mA。总电流消耗也取决于通过 I/O 引脚加入的电流数量。第 3 章列出了 ATmega168、ATmega328、ATmega1280、ATmega2560、ATmega32U4 微控制器的特定值。

2.11　更多信息

第 3 章将介绍 Arduino 开发板上 3 种 AVR MCU 使用的引脚，第 4 章将讲解如何把 AVR MCU 引脚映射到各种 Arduino 开发板的 I/O 引脚。

每种 AVR MCU 可能会有不同的内部外围电路组合，在一些情况下，不同类型的功能可能相差很小。若想了解关于 AVR MCU 的权威信息，请访问 Atmel 官方站点阅读相关文档（http://www.atmel.com）。

第3章

Arduino专用AVR微控制器

本章介绍各种 AVR 器件的技术参数，这些 8 位的 AVR 设备应用于各种型号的 Arduino 开发板。第 2 章已经介绍过 AVR 微控制器的基本功能，在此基础之上，本章将继续讲解相关内容，但把重点放在 ATmega168/328、ATmega1280/2560、ATmega32U4 微控制器上。

从使用 IDE 为 Arduino 进行编程的角度看，微控制器是对底层实际 AVR 设备的简单抽象。并且，执行操作的代码相当简单，比如配置一个输出引脚以产生 PWM 信号，或者在内部将模拟电压导入内置 ADC。控制寄存器内部地址及其控制位都已预定义，所以程序编写者无需担心底层细节，把主要精力放在编写控制逻辑上即可。

其实，Arduino 开发板只是一个 AVR 芯片的载体，所以 Arduino 的电气特性多半指处理器的电气特性。芯片引脚直接连接 Arduino 开发板边缘的针型端子或焊盘。芯片与开发板的连接点之间不存在缓冲或电平位移。

写作本书之时，Arduino 使用 5 种基本类型的 ATmega 微控制器和 3 种变型，总共 8 种部件型号，如表 3-1 所示。各种 AVR 设备的主要区别在于板载闪存的数量、最大时钟频率、芯片上 I/O 引脚的数目，以及可用的内部外围电路。ATmega32U4 设备也有一个内置的 USB 接口，这样就无需使用另一个部件处理 USB 通信。所有设备都使用相同的 CPU 指令集。

表3-1：Arduino产品使用的AVR微控制器

微控制器	闪存	I/O引脚（最多）	备注
ATmega168	16 kB	23	时钟频率 20 MHz
ATmega168V	16 kB	23	时钟频率 10 MHz
ATmega328	32 kB	23	时钟频率 20 MHz
ATmega328P	32 kB	23	时钟频率 20 MHz，微型电源
ATmega328V	32 kB	23	时钟频率 10 MHz
ATmega1280	128 kB	86	时钟频率 16 MHz
ATmega2560	256 kB	86	时钟频率 16 MHz
ATmega32U4	32 kB	26	时钟频率 16 MHz

这一固有的兼容性级别意味着，为 Arduino Diecimila 编写的程序可以不做任何修改地进行编译，并运行在 Uno 开发板上。Diecimila 与 Uno 之间最重要的区别是前者使用 ATmega168，而后者则基于 ATmega328P。正如表 3-1 所示，ATmega328P 板载闪存是 ATmega168 的两倍。因此，如果为 Uno 编写的程序在运行时需要使用更大的内存空间，那么它很有可能无法在 Diecimila 上正常运行。

3.1 ATmega168/328

从根本上说，ATmega168 与 ATmega328 是一样的芯片，只是带有不同大小的板载内存。图 3-1 显示的是 ATmega168 与 ATmega328 设备的框图。

3.1.1 内存

ATmega328 中的每种内存都是 ATmega168 的两倍，如表 3-2 所示。除此之外，两者完全相同。

表3-2：ATmega168/328板载内存

	ATmega168	ATmega328
程序内存	16 kB	32 kB
EEPROM	512 B	1 kB
RAM	1 kB	2 kB

3.1.2 特性

ATmega168/328 拥有如下特性：

- 支持片上启动程序的系统内编程（in-system programming by on-chip boot program）
- 2 个 8 位定时器 / 计数器，带有独立预分频器和比较模式
- 1 个 16 位定时器 / 计数器，带有独立预分频器、比较模式和捕获模式
- 实时计数器，配有独立振荡器
- 6 个 PWM 通道

- 6 个或 8 个通道的 10 位 ADC（取决于封装类型）
- USART
- 主从 SPI 串行接口
- 双线串行接口（兼容 Philips I2C）
- 可编程看门狗定时器
- 模拟比较器
- 23 个可编程 I/O 线路

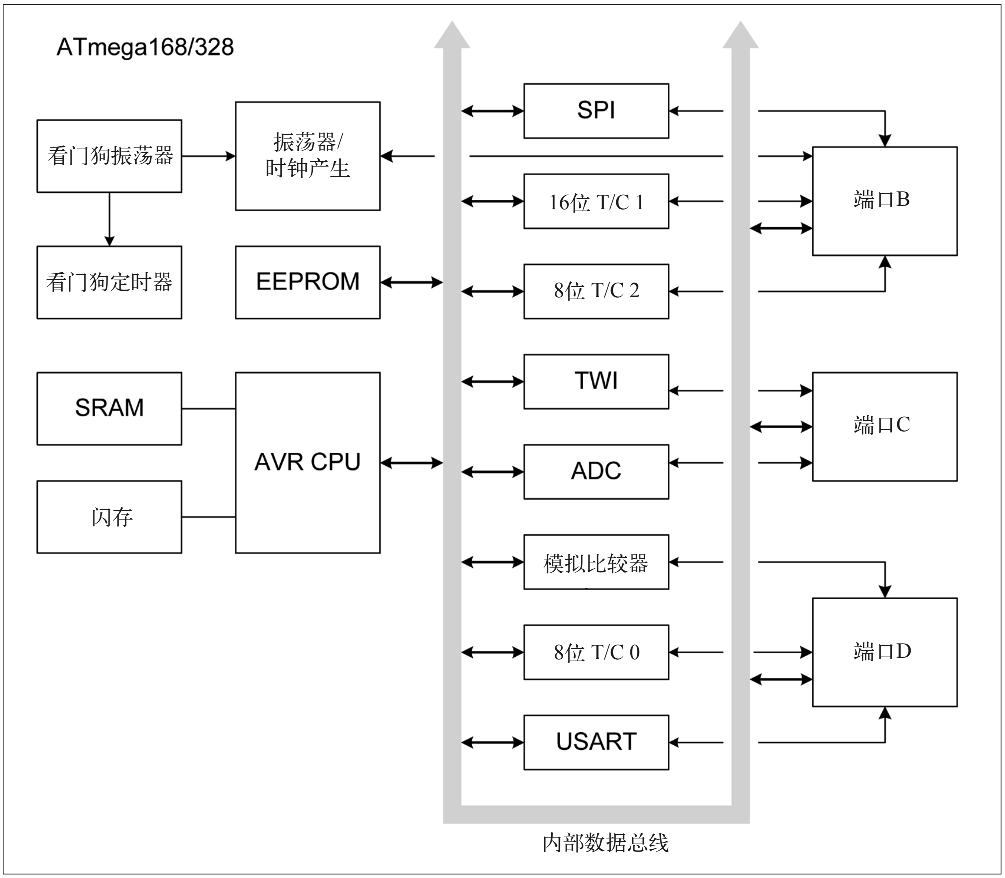


图 3-1：ATmega168/328 微控制器内部框图

3.1.3 封装

ATmega168 与 ATmega328 有 4 种不同的封装类型可用，分别为 28-pin DIP、28-pin MLF 表面贴装、32-pin TQFP 表面贴装、32-pin MLF 表面贴装。其中，28-pin DIP 是 Arduino 开发板最常用的封装方式，但 Uno SMD 除外，它使用 32-pin 表面贴装封装方式。本部分重点讲解 ATmega168 的 PDIP 版本。

3.1.4 端口

ATmega168/328 设计有 3 种类型的端口，分别为 B、C、D。其中，端口 B 与 D 是 8 位端口，端口 C 有 6 个可用的引脚，它们可用作 ADC 输入。PC4 与 PC5 也被连接到 TWI 逻辑，提供 SCL 和 SDA I2C 兼容的信号（时钟和数据）。此外，还要注意 PC6 通常被用作 RESET 输入。ATmega168/328 中没有 PC7。同时请注意，ATmega168/328 中没有端口 A。

每个端口提供双向独立的数字 I/O，并带有可编程的内部上拉电阻。上拉电阻的 on/off 状态借由控制寄存器中特定的端口引脚位进行选择。

端口输出缓冲拥有对称的驱动特性，并带有汇和源的能力（sink and source capability）。作为输入，如果激活内部上拉电阻，那么从外部拉低的端口引脚将拉电流。当重置条件激活时，即使时钟没有运行，端口引脚也会被设置成三态（tri-state）模式（高阻抗状态）。

3.1.5 引脚电路

图 3-2 显示了 28-pin DIP 封装的引脚电路。有关表面贴装封装引脚信息的更多内容，请参考 Atmel 技术文档（http://bit.ly/atmel-docs）。

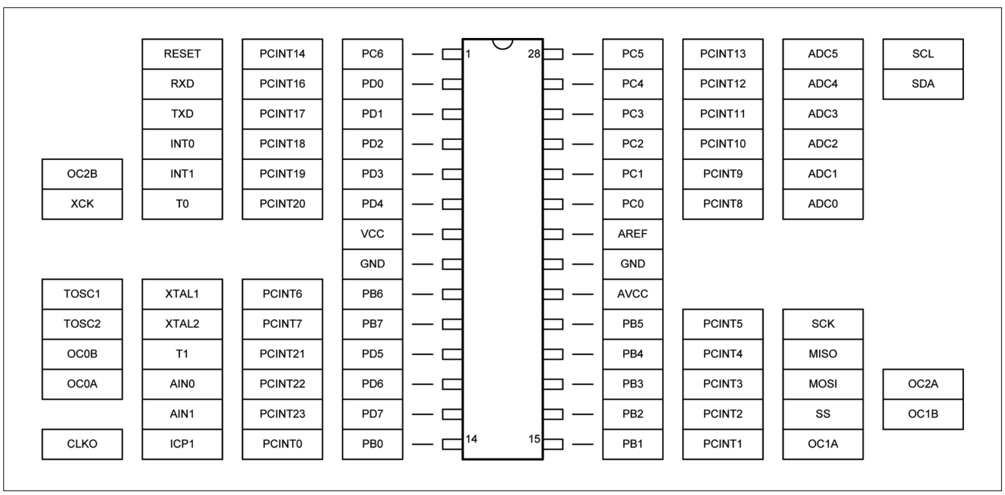


图 3-2：ATmega168/328 微控制器 DIP 封装的引脚电路

3.1.6 模拟比较器输入

图 3-3 显示使用 PDIP 封装的 ATmega168 或 ATmega328 的 AIN0 与 AIN1 引脚的位置。请注意，AIN0 与 OC0A 定时器 / 计数器输出（PD6）共享一个引脚。所以，如果 PD6 被用作 PWM 输出，它就不能被用作 AIN0 输入，除非每次需要修改其功能时进行重新配置。

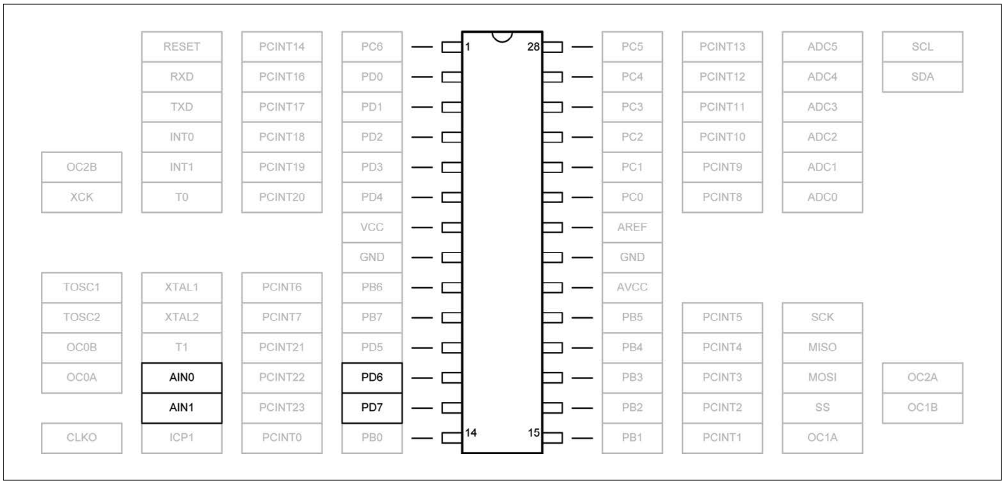


图 3-3：ATmega168/328 微控制器模拟比较器输入引脚

3.1.7 模拟输入

除了用于 TWI 串行通信的 SCL 与 SDA 引脚之外，模拟输入引脚也可以用作独立的数字 I/O，或者用作模拟输入，不与 AVR 其他周边电路发生冲突。图 3-4 显示 ATmega168 的 PDIP 封装的引脚，它们被用作模拟输入。

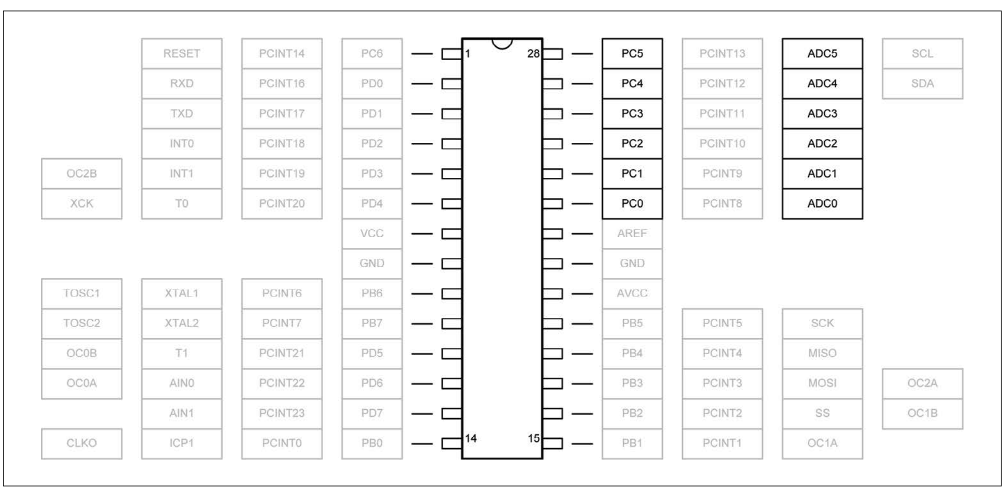


图 3-4：ATmega168/328 微控制器 ADC 输入引脚

3.1.8 串行接口

图 3-5 显示 ATmega168（PDIP 封装）的 I/O 引脚，串口电路会使用它们。由于没有串行电路共享端口引脚，所以 3 种类型都可以使用，不会发生端口冲突。

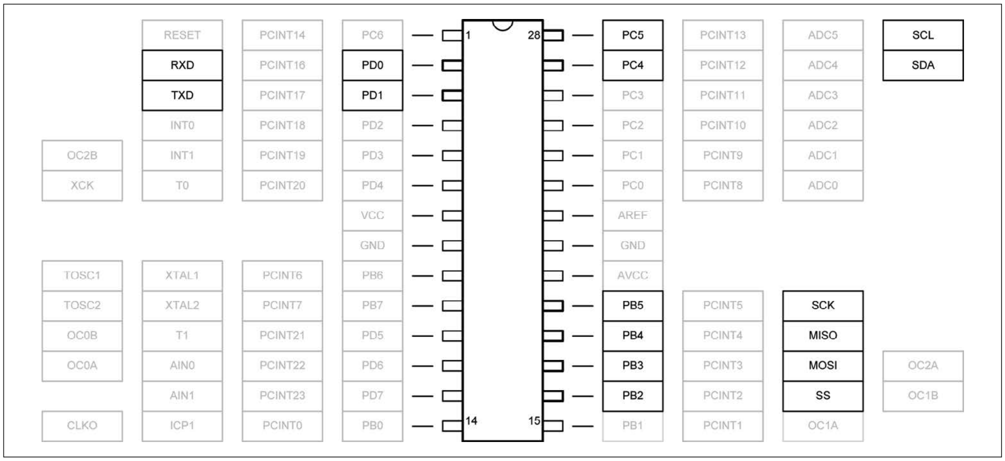


图 3-5：ATmega168/328 微控制器串行 I/O 引脚

3.1.9　定时器/时钟I/O

ATmega168/328 的内部定时器 / 计数器逻辑相当复杂，这反映在芯片的引脚分配上，如图 3-6 所示。请注意，OCxn 引脚（OC0A、OC0B、OC1A、OC1B、OC2A、OC2B）可以用作 PWM 输出，并且在 Arduino 开发板上都被打上了标签。还要注意 T1 与 OSC0B 共用相同的引脚（PD5），但也可以使用其他具有 PWM 能力的输出，且不会与其他定时器 / 计数器电路发生冲突。

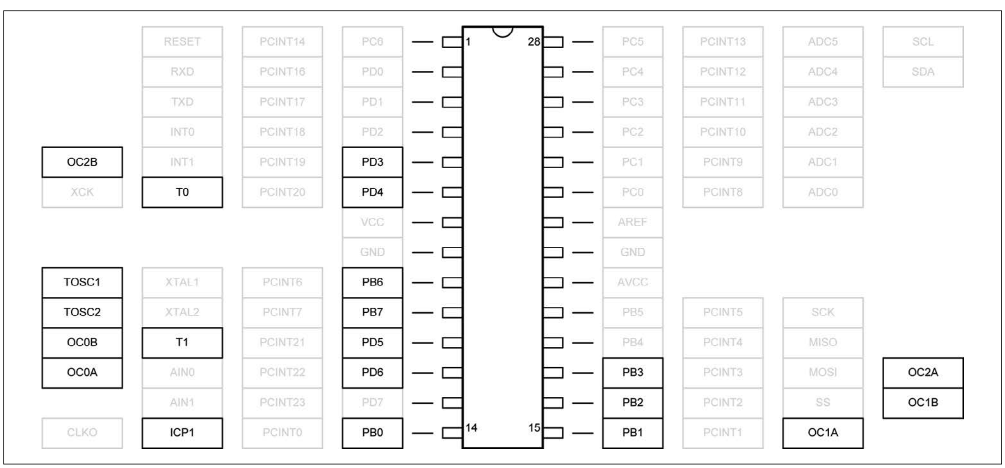


图 3-6：ATmega168/328 微控制器定时器 / 计数器引脚

3.1.10　外部中断

端口 D 的 PD2 与 PD3 引脚特意被用作外部中断输入。PCINT0~PCINT23 的任意引脚都可以被用作外部中断输入，并且不会干扰其他已分配的功能（请参考 2.8 节了解如何使用这

些中断）。图 3-7 显示 ATmega168/328 设备中可用的外部中断输入。

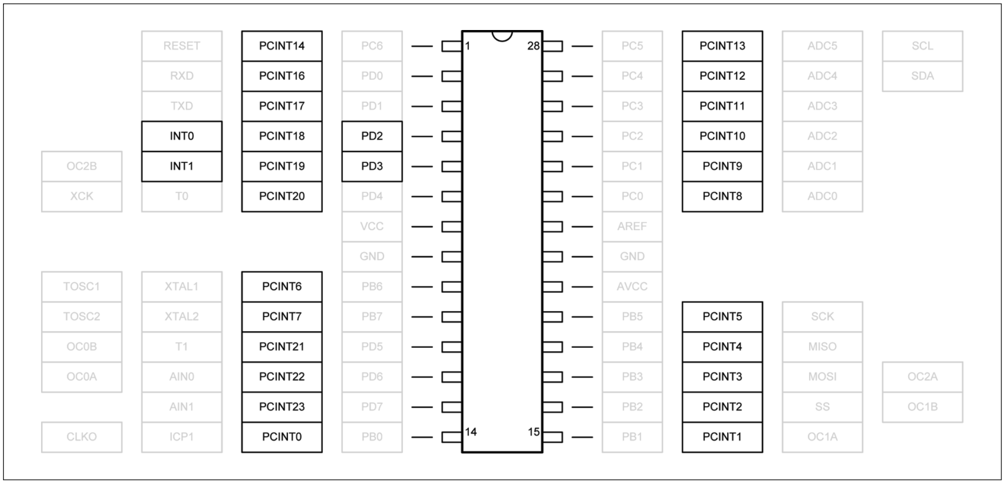


图 3-7：ATmega168/328 微控制器中断输入引脚

3.1.11 Arduino引脚分配

Arduino Diecimila、Duemilanove、Uno、Ethernet 开发板使用相同的基本布局方式（第 4 章），如图 3-8 所示。这些 Arduino 开发板针对其上的引脚插口头采用统一命名，这些命名基于安装在 PCB 上的 ATmega168 或 ATmega328 的各个引脚功能。其中，主要强调的是最常用的功能，比如独立的数字 I/O、模拟输入，以及 AVR 微控制器的 PWM 输出能力。

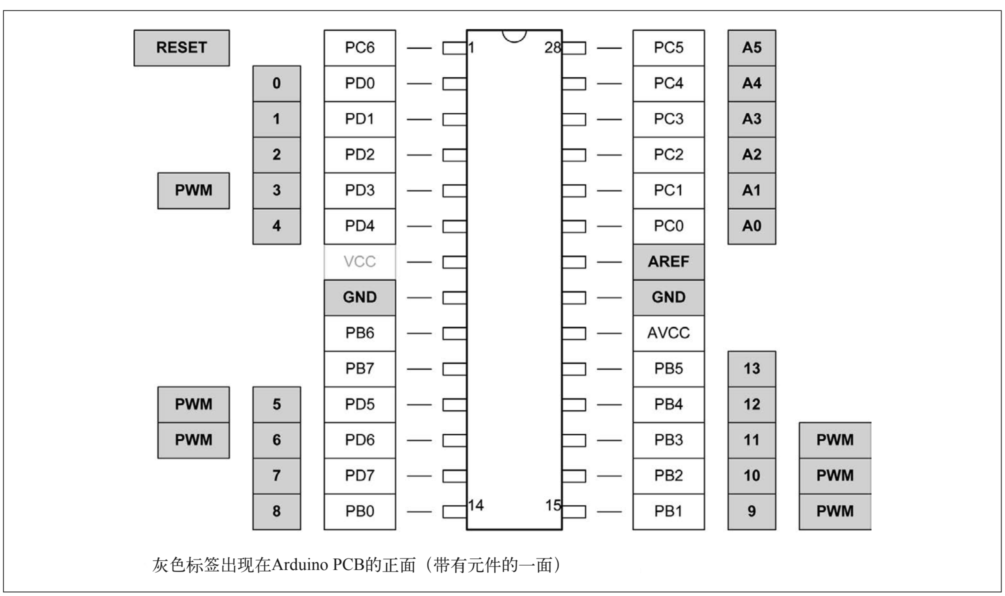


图 3-8：ATmega168/328 引脚的 Arduino 标签

3.1.12 基本电气特性

表 3-3 显示了 ATmega168/328 微控制器的一些基本电气特性，重点强调的是电力消耗。

表3-3：ATmega168/328电力消耗特性

Device	Parameter	Condition	VCC	Typical	Max
ATmega168	Power supply current	Active 8 MHz	5 V	4.2 mA	12 mA
		Idle 8 MHz	5 V	0.9 mA	5.5 mA
	Power-save mode	32 MHz TOSC	1.8 V	0.75 µA	
		32 MHz TOSC	5 V	0.83 µA	
	Power-down mode	WDT enabled	3 V	4.1 µA	15 µA
		WDT disabled	3 V	0.1 µA	2 µA
ATmega328	Power supply current	Active 8 MHz	5 V	5.2 mA	12 mA
		Idle 8 MHz	5 V	1.2 mA	5.5 mA
	Power-save mode	32 MHz TOSC	1.8 V	0.8 µA	
		32 MHz TOSC	5 V	0.9 µA	
	Power-down mode	WDT enabled	3 V	4.2 µA	15 µA
		WDT disabled	3 V	0.1 µA	2 µA

当 VCC 在 1.8 V~2.4 V 时，I/O 引脚上的低电平输入被定义为在 −0.5 V~0.2 VCC 的一个电压。当 VCC 在 2.4 V~5.5 V 时，低电平输入被定义为 −0.5 V~0.3 VCC 的一个电压。

当 VCC 在 1.8 V~2.4 V 时，高电平输入被定义为在 0.7 VCC~VCC+0.5 V 的一个电压。当 VCC 在 2.4 V~5.5 V 时，高电平输入被定义为 0.6 VCC~VCC+0.5 V 的一个电压。

3.2 ATmega1280/ATmega2560

正如 ATmega168 与 ATmega328 的区别一样，ATmega1280 与 ATmega2560 的主要差别在于板载内存的大小。除此之外，两者完全一样。图 3-9 是对 ATmega1280 与 ATmega2560 设备进行简化之后的框图。

为清晰起见，图 3-9 省略了一些小细节，但必要部分都已包含其中。查看 Atmel 提供的有关 ATmega1280 与 ATmega2560 的文档，可以看到更详细的框图。此外，请注意图 3-9 显示了可用的内部电路，采用的是 100-pin 封装。64-pin 封装支持的是它的一个子集。

3.2.1 内存

ATmega2560 的每种内存都是 ATmega1280 的两倍，是 ATmega328 MCU 闪存大小的 8 倍。表 3-4 列出了可用的内存。除此之外，ATmega1280 与 ATmega2560 采用了一样的封装类型，它们完全相同。

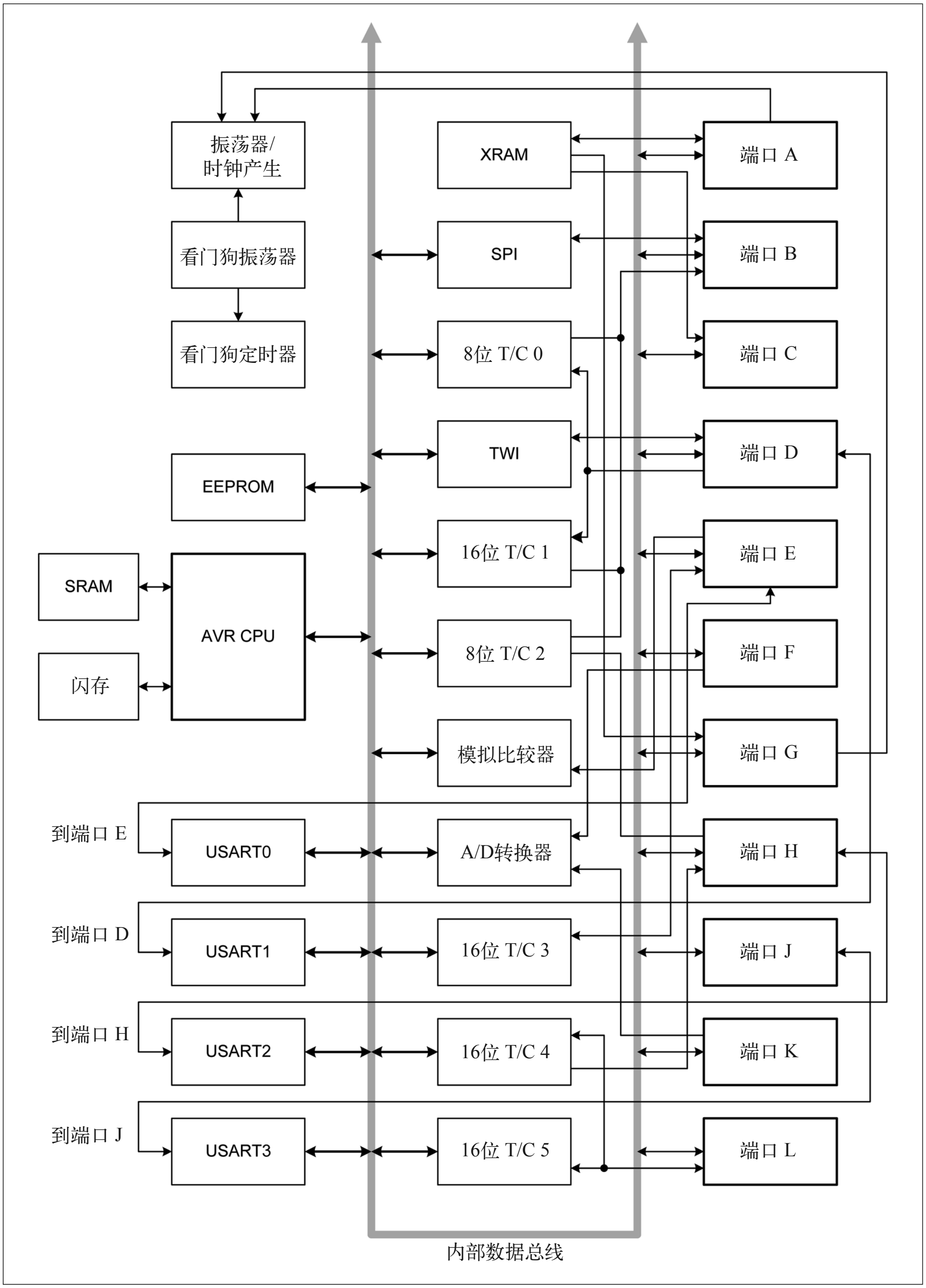


图 3-9：ATmega1280/2560 微控制器内部框图

表3-4：ATmega1280/2560板载内存

	ATmega1280	ATmega2560
程序闪存	128 kB	256 kB
EEPROM	4 kB	4 kB
RAM	8 kB	8 kB

3.2.2 特性

ATmega1280/2560 具有如下一些特征：

- 支持片上启动程序的系统内编程
- 2 个 8 位定时器 / 计数器，带有独立预分频器和比较模式
- 4 个 16 位定时器 / 计数器，带有独立预分频器、比较模式和捕获模式
- 实时计数器，配有独立振荡器
- 12 个 PWM 通道
- 输出比较调制器（output compare modulator）
- 6 个通道，10 位 ADC
- 4 个 USART 电路
- 主从 SPI 串行接口
- 双线串行接口（兼容 Philips I2C）
- 可编程看门狗定时器
- 模拟比较器
- 86 个可编程 I/O 线路

3.2.3 封装

ATmega1280 与 ATmega2560 设备可以采用 100-pin 的 TQFP 封装、100-pin 的 BGA（球栅阵列）封装、64-pin 的 TQFP 封装。图 3-10 显示相对大小，以及 3 种可用封装类型的引脚间距。请注意，只有 100-pin 的封装才提供图 3-9 中显示的所有功能。

Arduino Mega 与 Mega2560 开发板都采用 100-pin 的 TQFP 封装方式。

3.2.4 端口

ATmega1280 与 ATmega2560 的 100-pin 封装版本拥有 11 个端口，标注为 A~L。请注意，其中不包含 I 端口，因为字母 I 很容易被误认为数字 1。

端口 A、B、C、D、E 是双向 8 位端口。与其他端口相比，端口 B 具有更好的驱动性能。端口 F 与 K 用作内部 A/D 转换器的输入，但也可以用作双向端口。端口 G 是 6 位端口，端口 H、J、L 是 8 位双向接口。每个端口都提供双向离散的数字 I/O，并带有可编程的内部上拉电阻。上拉电阻的 on/off 状态借由控制寄存器中特定的端口引脚位进行选择。

端口输出缓冲拥有对称的驱动特性，并带有汇和源的能力。作为输入，如果激活内部上拉电阻，那么从外部拉低的端口引脚将拉电流。当重置条件激活时，端口引脚也会被设置成

三态模式（高阻抗状态）。

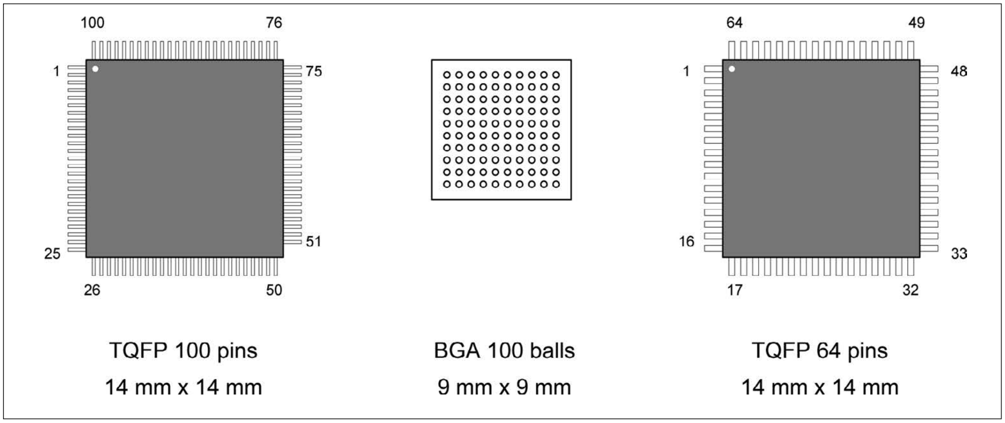


图 3-10：ATmega1280/2560 封装

3.2.5 引脚功能

本部分的框图指的是 ATmega1280/2560 设备的 100-pin 封装版本。关于 BGA 与 64-pin 封装的更多细节，请参考 Atmel 的相关技术文档。

3.2.6 模拟比较器输入

像其他更小的 ATmega168/328 设备一样，ATmega1280/2560 拥有两个模拟比较器输入，相关细节如表 3-5 与图 3-11 所示。

表3-5：模拟比较器输入

引脚	端口	功能
4	PE2	AIN0
5	PE3	AIN1

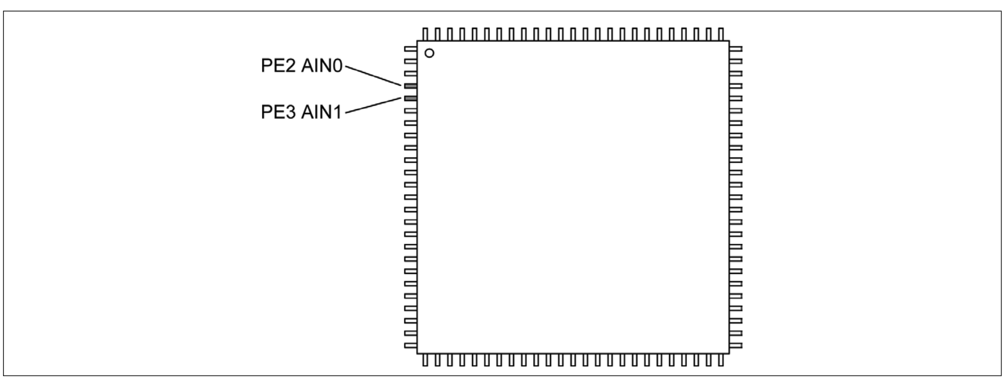


图 3-11：ATmega1280/2560 模拟比较器输入引脚

3.2.7 模拟输入

ATmega1280/2560 设备支持多达 16 个 A/D 的转换器输入。这些输入位于端口 F 与 K，并被连接到 82~97（100-pin 封装）的引脚上（见表 3-6）。端口 F 也有 TCK、TMS、TDI、TDO 等其他用途，端口 K 的引脚被连接到 PCINT16~PCINT23 中断端口。物理引脚的位置如图 3-12 所示。

表3-6：模拟输入

引脚	端口	功能	引脚	端口	功能
82	PK7	ADC15	90	PF7	ADC7
83	PK6	ADC14	91	PF6	ADC6
84	PK5	ADC13	92	PF5	ADC5
85	PK4	ADC12	93	PF4	ADC4
86	PK3	ADC11	94	PF3	ADC3
87	PK2	ADC10	95	PF2	ADC2
88	PK1	ADC9	96	PF1	ADC1
89	PK0	ADC8	97	PF0	ADC0

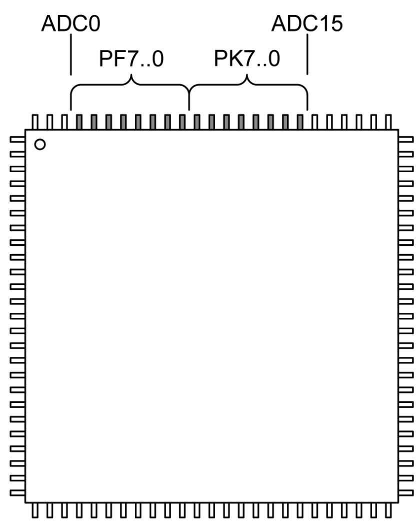


图 3-12：ATmega1280/2560 ADC 输入引脚

3.2.8 串行接口

ATmega1280/2560 设备拥有 4 个内部 USART 电路，对外引出 4 对引脚，1 对用作 TXD，其他用作 RXD。SPI 接口在端口 B 上可用，涉及的引脚为 19~22（100-pin 封装）。通过引脚 43 与 44，双线接口（I2C）被连接到端口 D。表 3-7~ 表 3-12 列出了引脚分配情况。图 3-13 显示串行 I/O 引脚的位置（100-pin 封装）。

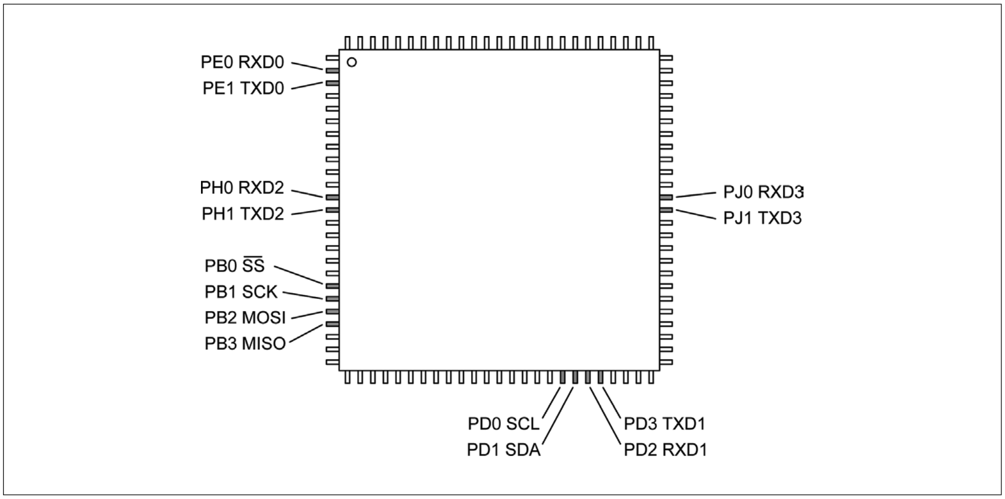


图 3-13：ATmega1280/2560 串行 I/O 引脚

表3-7：USART 0

引脚	端口	功能
2	PE0	RXD0
3	PE1	TXD0

表3-8：USART 1

引脚	端口	功能
45	PD2	RXD1
46	PD3	TXD1

表3-9：USART 2

引脚	端口	功能
12	PH0	RXD2
13	PH1	TXD2

表3-10：USART 4

引脚	端口	功能
63	PJ0	RXD3
64	PJ1	TXD3

表3-11：SPI

引脚	端口	功能
19	PB0	SS (active low)
20	PB1	SCK
21	PB2	MOSI
22	PB3	MISO

表3-12：TWI

引脚	端口	功能
43	PD0	SCL
44	PD1	SDA

3.2.9　定时器/时钟I/O

ATmega1280/2560 设备中有 5 种定时器 / 计数器功能，如图 3-14 所示。表 3-13 列出了相应引脚。请注意，其中不含 T2 引脚。

表3-13：Atmega1280/2560定时器/计数器引脚

引脚	端口	功能	引脚	端口	功能
1	PG5	OC0B	50	PD7	T0
5	PE3	OC3A	15	PH3	OC4A
6	PE4	OC3B	16	PH4	OC4B
7	PE5	OC3C	17	PH5	OC4C
8	PE6	T3	18	PH6	OC2B
9	PE7	ICP3	27	PH7	T4
23	PB4	OC2A	35	PL0	ICP4
24	PB5	OC1A	36	PL1	ICP5
25	PB6	OC1B	37	PL2	T5
26	PB7	OC0A/OC1C	38	PL3	OC5A
47	PD4	ICP1	39	PL4	OC5B
49	PD6	T1	40	PL5	OC5C

3.2.10　外部中断

ATmega1280/2560 设备支持 8 种外部中断输入，此外，端口 B、J、K 的端口中断功能均可用。相关引脚在表 3-14 中列出。

表3-14：ATmega1280/2560中断引脚

引脚	端口	功能	引脚	端口	功能
6	PE4	INT4	43	PD0	INT0
7	PE5	INT5	44	PD1	INT1
8	PE6	INT6	45	PD2	INT2
9	PE7	INT7	46	PD3	INT3

3.2.11　Arduino引脚分配

Arduino Mega 与 Mega2560 开发板使用的布局方式将在第 4 章进行讲解。这些 Arduino 开发板针对其上的引脚插口头采用统一命名，这些命名基于安装于 PCB 上的 ATmega1280 或 ATmega2650 的各个引脚功能。其中，主要强调的是最常用的功能，比如独立的数字 I/O、模拟输入，以及 AVR 微控制器的 PWM 输出性能。

表 3-15 列出 Mega 或 Mega2560 PCB 上的引脚，以及到开发板上 AVR ATmega1280/2560 设备的连接。括号中的功能是外存地址引脚，并且使用波浪线（~）表示低电平有效（低电平时为真）信号。不同于小型 Arduino 开发板，Mega 开发板可以使用外部 SRAM。

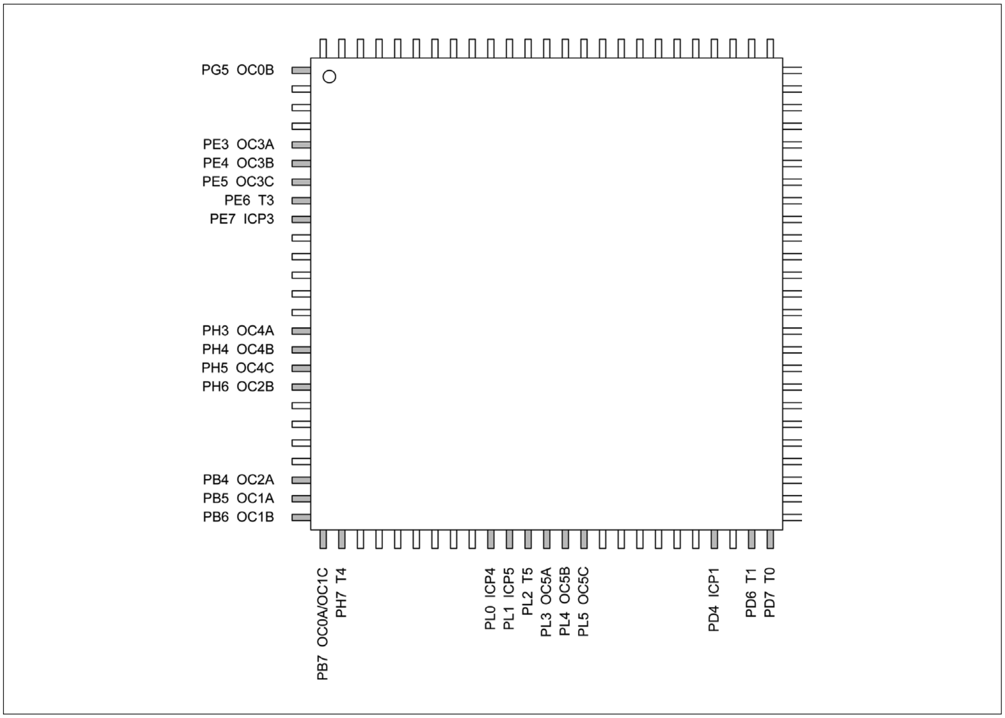


图 3-14：ATmega1280/2560 定时器 / 计数器引脚

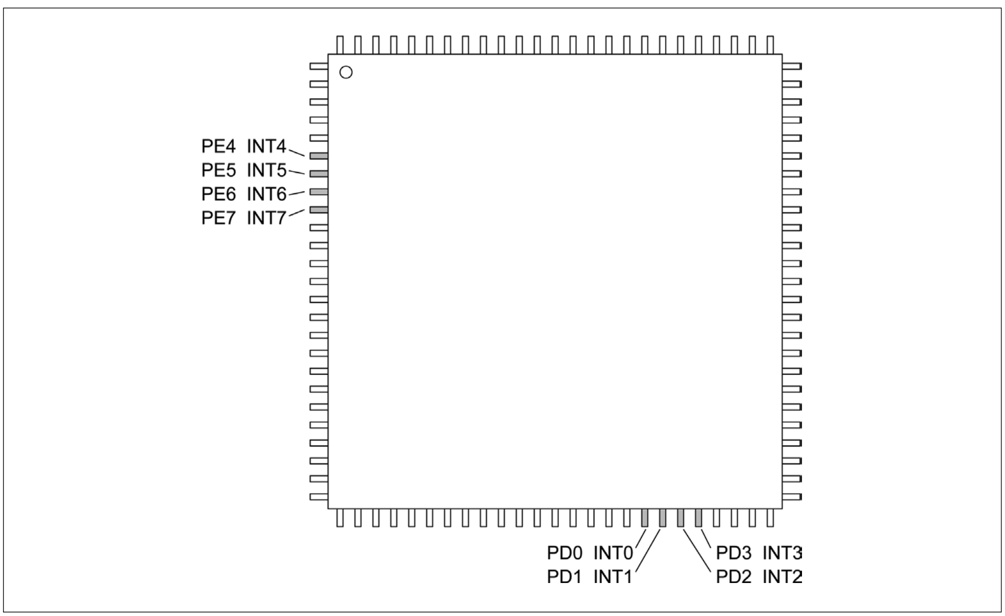


图 3-15：ATmega1280/2560 中断输入引脚

表3-15：Arduino Mega与Mega2560引脚分配

Mega开发板引脚	芯片引脚	功能	Mega开发板引脚	芯片引脚	功能
0	2	RXD0	35	55	I/O (A10)
1	3	TXD0	36	54	I/O (A9)
2	5	OC3B [PWM]	37	53	I/O (A8)
3	6	OC3C [PWM]	38	50	T0
4	1	OC0B [PWM]	39	70	I/O (ALE)
5	4	OC3A [PWM]	40	52	I/O (~RD)
6	15	OC4A [PWM]	41	51	I/O (~WR)
7	16	OC4B [PWM]	42	42	PL7
8	17	OC4C [PWM]	43	41	PL6
9	18	OC2B [PWM]	44	40	OC5C [PWM]
10	23	OC2A [PWM]	45	39	OC5B [PWM]
11	24	OC1A [PWM]	46	38	OC5A [PWM]
12	25	OC1B [PWM]	47	37	T5
13	26	OC0A [PWM]	48	36	ICP5
14	64	TXD3	49	35	ICP4
15	63	RXD3	50	22	MISO
16	13	TXD2	51	21	MOSI
17	12	RXD2	52	20	SCK
18	46	TXD1	53	19	~SS
19	45	RXD1	54	97	A0（逻辑输入）
20	44	SDA	55	96	A1（逻辑输入）
21	43	SCL	56	95	A2（逻辑输入）
22	78	I/O (AD0)	57	94	A3（逻辑输入）
23	77	I/O (AD1)	58	93	A4（逻辑输入）
24	76	I/O (AD2)	59	92	A5（逻辑输入）
25	75	I/O (AD3)	60	91	A6（逻辑输入）
26	74	I/O (AD4)	61	90	A7（逻辑输入）
27	73	I/O (AD5)	62	89	A8（逻辑输入）
28	72	I/O (AD6)	63	88	A9（逻辑输入）
29	71	I/O (AD7)	64	87	A10（逻辑输入）
30	60	I/O (A15)	65	86	A11（逻辑输入）
31	59	I/O (A14)	66	85	A12（逻辑输入）
32	58	I/O (A13)	67	84	A13（逻辑输入）
33	57	I/O (A12)	68	83	A14（逻辑输入）
34	56	I/O (A11)	69	82	A15（逻辑输入）

请注意，引脚 22~37 以及引脚 39、40、41 都可以用于访问外存。此外，它们也能被用作一般的独立数字 I/O 引脚。

3.2.12 电气特性

表 3-16 显示了 ATmega1280/2560 微控制器的一些基本电气特性，重点强调的是电力消耗。

表3-16：ATmega1280/2560电力消耗特性

设备	参数	条件	VCC	标准电流	最大电流
ATmega1280	电源电流	Active 8 MHz	5 V	10 mA	14 mA
ATmega2560	电源电流	Idle 8 MHz	5 V	2.7 mA	4 mA
both	省电模式	启用 WDT	3 V	<5 μA	15 μA
		关闭 WDT	3 V	<1 μA	7.5 μA

当 VCC 在 1.8 V~2.4 V 时，I/O 引脚上的低电平输入被定义为 −0.5 V~0.2 VCC 的一个电压。当 VCC 在 2.4 V~5.5 V 时，低电平输入被定义为 −0.5 V~0.3 VCC 的一个电压。

当 VCC 在 1.8 V~2.4 V 时，高电平输入被定义为在 0.7 VCC~VCC+0.5 V 的一个电压。当 VCC 在 2.4 V~5.5 V 时，高电平输入被定义为 0.6 VCC~VCC+0.5 V 的一个电压。

3.3 ATmega32U4

ATmega32U4 是 Atmel XMEGA 微控制器产品线的成员之一。它有 32 kB 程序闪存，2.5 kB 的 SRAM、1 kB 的 EEPROM。I/O 功能可以通过端口 B 与 F 进行访问。ATmega32U4 中不存在端口 A。图 3-16 是 ATmega32U4 主要内部组件的框图。

ATmega32U4 特色之一是带有一个集成的 USB 2.0 全速接口，因而不再需要独立的板外接口芯片。ATmega32U4 也有一个兼容 1149.1 的 JTAG 接口，用于片上调试。ATmega32U4 正常工作的电压范围是 2.7 V~5.5 V。

3.3.1 内存

如表 3-17 所示，ATmega32U4 拥有与 ATmega328 一样大小的闪存与 EEPROM，而 RAM 的大小为 2.5 kB，而非 2 kB。

表3-17：ATmega32U4板载内存

引脚	端口
程序闪存	32 kB
EEPROM	1 kB
RAM	2.5 kB

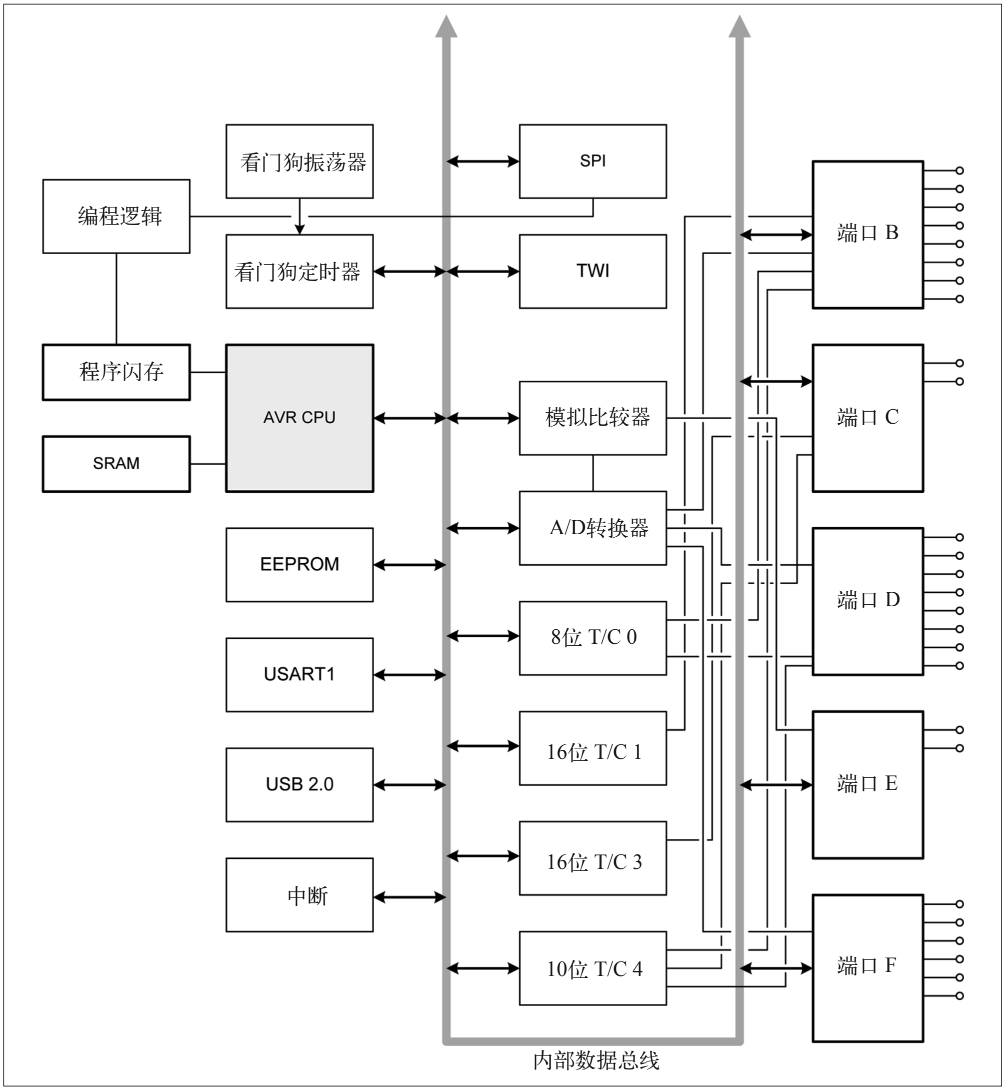


图 3-16：ATmega32U4 微控制器内部结构框图

3.3.2 特性

ATmega32U4 拥有如下特性：

- 支持片上启动程序的系统内编程
- 1 个 8 位定时器 / 计数器，带有独立预分频器和比较模式
- 2 个 16 位定时器 / 计数器，带有独立预分频器、比较模式和捕获模式
- 1 个 10 位高速定时器 / 计数器，带有锁相环（PLL）和比较模式
- 4 个 8 位 PWM 通道

- 4 个 PWM 通道，拥有 2~16 位可编程分辨率
- 6 个高速 PWM 通道，拥有 2~11 位可编程分辨率
- 输出比较调制器
- 12 通道，10 位 ADC
- USART 功能，带有 CTS/RTS 握手
- 主从 SPI 串行接口
- 双线串行接口（兼容 Philips I2C）
- 可编程看门狗定时器
- 模拟比较器
- 片上温度传感器
- 26 个可编程 I/O 线路

3.3.3 封装

ATmega32U4 可以采用 TQFP44 或 QFN44 表面贴片封装方式。图 3-17 中显示的引脚名称对于两种封装都是一样的。

3.3.4 端口

ATmega32U4 拥有 5 个端口，标记为 B~F。在 44-pin 的 QFN/TQFP 表面贴装封装中，只有端口 B 与 D 的全部 8 位出现在封装引脚上。端口 C、E、F 在内部被作为 8 位寄存器，但只有端口 C 的 PC6 与 PC7 位可以在外部使用。对于端口 E，只有 PE2 与 PE6 两个位可以在外部使用；对于端口 F，PF2 与 PF3 两个位不出现在封装引脚上。

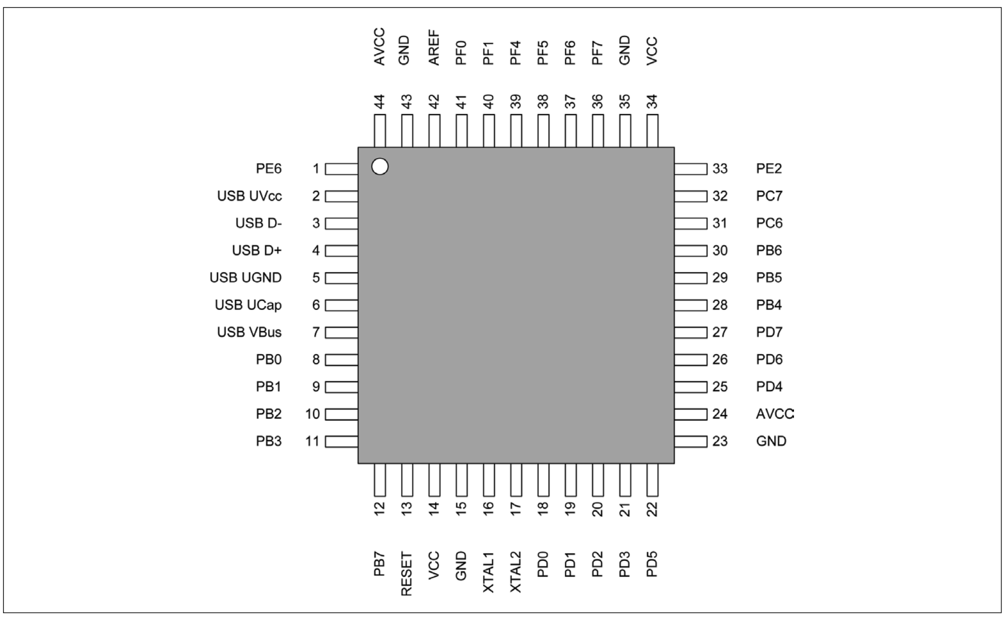


图 3-17：ATmega32U4 微控制器 44-pin 封装

此外，各种外围功能端口 B、C、D、E、F 都是双向离散的数字 I/O 端口。每个端口提供双向独立的数字 I/O，并带有可编程的内部上拉电阻。上拉电阻的 on/off 状态借由控制寄存器中特定的端口引脚位进行选择。

端口输出缓冲拥有对称的驱动特性，并带有汇和源的能力。作为输入，如果激活内部上拉电阻，那么从外部拉低的端口引脚将拉电流。当重置条件激活时，端口引脚会被设置成三态模式（高阻抗状态）。

3.3.5 引脚功能

表 3-18 列出了采用 44-pin 封装的 ATmega32U4 的引脚分配情况。在 Arduino Leonardo 开发板上，关于芯片引脚到排针的映射可参考 3.3.13 节。

表3-18：ATmega32U4 TQFP/QFN-44引脚分配

引脚	功能	端口	引脚	功能	端口	引脚	功能	端口
1	INT6/AIN0	PE6	16	XTAL2	n/a	31	OC3A/OC4A	PC6
2	USB UVCC	n/a	17	XTAL1	n/a	32	ICP3/CLK0/OC4A	PC7
3	USB D–	n/a	18	OC0B/SCL/INT0	PD0	33	HWB	PE2
4	USB D+	n/a	19	SDA/INT1	PD1	34	VCC	n/a
5	USB UGnd	n/a	20	RXD1/INT2	PD2	35	GND	n/a
6	USB UCap	n/a	21	TXD1/INT3	PD3	36	ADC7/TDI	PF7
7	USB VBus	n/a	22	XCK1/CTS	PD5	37	ADC6/TD0	PF6
8	PCINT0/SS	PB0	23	GND	n/a	38	ADC5/TMS	PF5
9	PCINT1/SCLK	PB1	24	AVCC	n/a	39	ADC4/TCK	PF4
10	PDI/PCINT2/MOSI	PB2	25	ICP1/ADC8	PD4	40	ADC1	PF1
11	PDO/PCINT3/MISO	PB3	26	T1/OC4D/ADC9	PD6	41	ADC0	PF0
12	PCINT7/OC0A/OC1C/RTS	PB7	27	T0/OC4D/ADC10	PD7	42	AREF	n/a
13	RESET	n/a	28	PCINT4/ADC11	PB4	43	GND	n/a
14	VCC	n/a	29	PCINT5/OC1A/OC4B/ADC12	PB5	44	AVCC	n/a

3.3.6 模拟比较器输入

在 ATmega32U4 中，模拟比较器只有一个外部输入。AIN0 位于引脚 1。模拟比较器的其他输入来自于片上 ADC 的输入多路信号选择器。内部电路与 2.5 节显示的完全一样，但不带有 AIN1 引脚连接。

3.3.7 模拟输入

ATmega32U4 提供 12 个 A/D 转换器输入，如表 3-19 与图 3-18 所示。这些输入位于端口 B、D、F。请注意，ADC2 与 ADC3 没有外部引脚。

表3-19：ATmega32U4 ADC输入

引脚	端口	功能	引脚	端口	功能	引脚	端口	功能
41	PF0	ADC0	37	PF6	ADC6	27	PD7	ADC10
40	PF1	ADC1	36	PF7	ADC7	28	PB4	ADC11
39	PF4	ADC4	25	PD4	ADC8	29	PB5	ADC12
38	PF5	ADC5	26	PD6	ADC9	30	PB6	ADC13

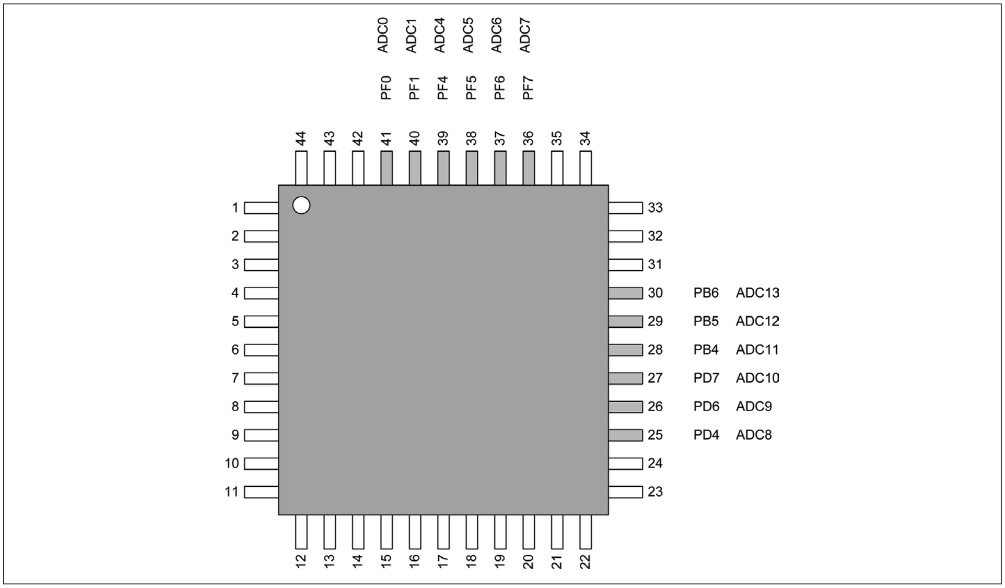


图 3-18：ATmega132U4 ADC 输入引脚

3.3.8 串行接口

ATmega32U4 拥有一个内部 USART 电路（带有硬件握手线路）、一个 SPI 接口和一个与 I2C 兼容的双向接口（TWI）（表 3-20 ~ 表 3-22）。USB 接口在后面图 3-22 中讲解。引脚分配如图 3-19 所示。

表3-20：USART

引脚	端口	功能	引脚	端口	功能
20	PD2	RXD1	22	PD5	CTS
21	PD3	TXD1	12	PB7	RTS

表3-21：SPI

引脚	端口	功能	引脚	端口	功能
8	PB0	SS	10	PB2	MOSI
9	PB1	SCLK	11	PB3	MISO

表3-22：TWI

引脚	端口	功能
18	PD0	SCL
19	PD1	SDA

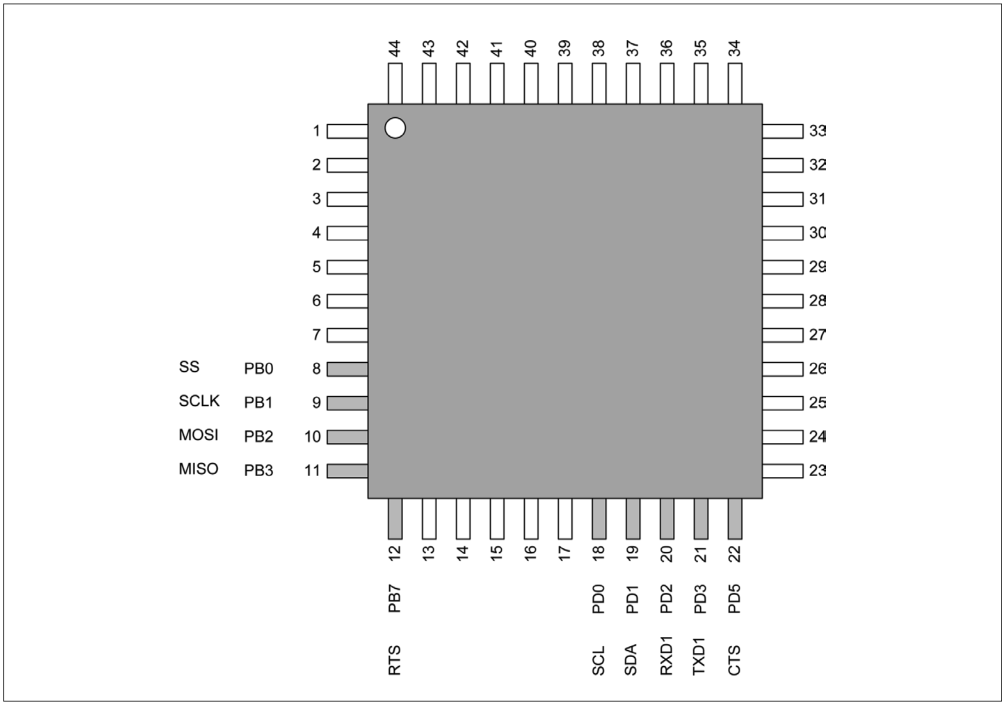


图 3-19：ATmega32U4 串行 I/O 引脚

3.3.9　定时器/时钟I/O

ATmega32U4 拥有 4 个片上定时器 / 计数器电路。它们由一个片上 8 位定时器 / 计数器、两个 16 位定时器 / 计数器（带有独立预分频器和一个比较模式）、一个高速 10 位定时器 / 计数器（带有锁相环 PLL 和一个比较模式）组成。请注意，16 位定时器 / 计数器也支持捕获模式。

定时器编号为 0~4，但不存在定时器 / 计数器 2。

定时器 / 计数器 0 的 OC1A、OC1B、T0 引脚与定时器 / 计数器 4（高速定时器 / 计数器）共用。与之类似，定时器 / 计数器 1 与 3 的引脚也与定时器 / 计数器 4 的可用引脚共用。

表 3-23 列出了 ATmega32U4 MCU 的定时器 / 时钟电路的引脚分配情况。波浪线（~）表示低电平有效（低电平时为真）引脚。ATmega32U4 中不存在 T3 或 T4 引脚。只有定时器 / 计数器 0 与 1 拥有可用的 T0 与 T1 引脚。引脚分配如图 3-20 所示。

表3-23：定时器/时钟引脚

引脚	端口	功能	引脚	端口	功能
12	PB7	OC0A/OC1C	29	PB5	OC1A/~OC4B
18	PD0	OC0B	30	PB6	OC1B/OC4B
25	PD4	ICP1	31	PC6	OC3A/OC4A
26	PD6	T1/~OC4D	32	PC7	ICP3/~OC4A

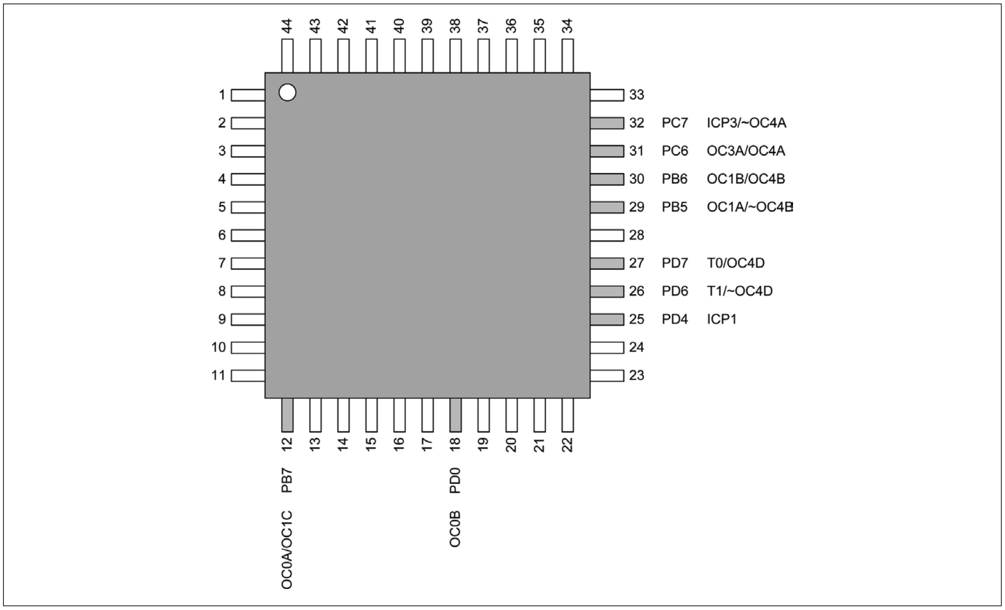


图 3-20：ATmega32U4 定时器 / 计数器引脚

3.3.10　外部中断

ATmega32U4 MCU 的中断引脚分配如表 3-24 与图 3-21 所示。端口 D 的 PD0~PD3 引脚以及端口 E 的 PE6 引脚被特意用作外部中断输入。PCINT0~PCINT7 的任意引脚也可以被用作外部中断输入，不会受到其他功能的干扰（参考 2.8 节了解如何使用中断）。

表3-24：外部中断引脚

引脚	端口	功能	引脚	端口	功能
8	PB0	PCINT0	12	PB7	PCINT7
9	PB1	PCINT1	18	PD0	INT0
10	PB2	PCINT2	19	PD1	INT1
11	PB3	PCINT3	20	PD2	INT2
28	PB4	PCINT4	21	PD3	INT3
29	PB5	PCINT5	1	PE6	INT6

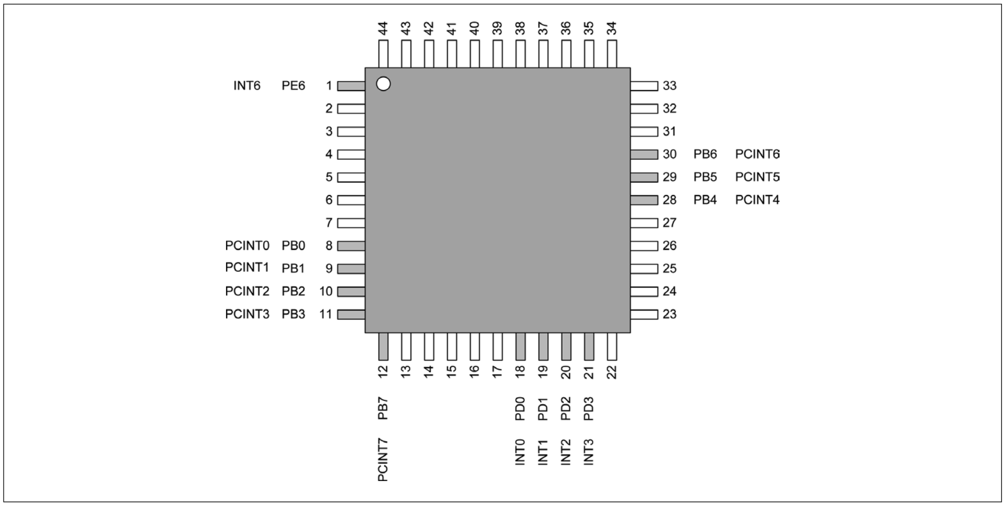


图 3-21：ATmega32U4 外部中断引脚

3.3.11　USB 2.0接口

ATmega32U4 集成了一个全速 USB 2.0 接口，但它不能是 USB 主机，只能是一个 USB 设备。它提供了多个内部端点，带有可配置的 FIFO 缓存。片上 PLL 为 USB 接口产生 48 MHz 的时钟，并且可以配置 PLL 输入使用外部振荡器、外部时钟源或者内部 RC 时钟源。来自 PLL 的 48 MHz 输出用于产生 12 MHz 全速或 1.5 MHz 低速时钟。

请注意，ATmega32U4 的 USB 引脚并未关联到某个端口，只与内部 USB 逻辑与稳压电路有关。USB I/O 引脚分配如表 3-25 与图 3-22 所示。

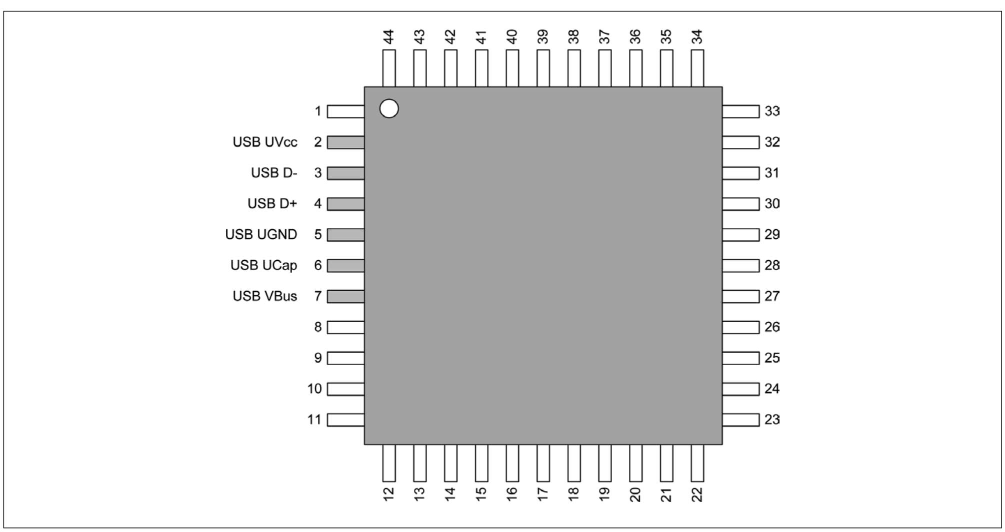


图 3-22：ATmega32U4 USB 引脚

表3-25：USB I/O

引脚	功能	引脚	功能
2	USB UVCC	5	USB UGnd
3	USB D–	6	USB UCap
4	USB D+	7	USB VBus

3.3.12 电气特性

表 3-26 显示 ATmega32U4 基本的电气参数（最大值），表 3-27 显示基本的电流消耗参数。

表3-26：ATmega32U4最大额定值

参数	取值	单位
I/O 引脚电压	–0.5 ~ VCC + 0.5	V
重置引脚电压	–0.5 ~ +13.0	V
VBUS 引脚电压	–0.5 ~ +6.0	V
设备 VCC	6.0	V
I/O 引脚直流	40.0	mA
VCC 电流	200.0	mA

表3-27：ATmega32U4电力消耗特性

参数	条件	VCC	标准值	最大值
电源电流	Active 8 MHz	5 V	10 mA	15 mA
	Idle 8 MHz	5 V		6 mA
省电模式	WDT enabled	3 V	<10 μA	12 μA
	WDT disabled	3 V	1 μA	5 μA

3.3.13 Arduino引脚分配

ArduinoLeonardo 开发板使用扩展的基准线布局，包括额外引脚。USB 接口引脚被直接连接到 PCB 上的 USB 连接器，USB 信号不会出现在 PCB 侧边的引脚插口头上。关于 Leonardo 开发板的布局将在第 4 章进行介绍。

表 3-28 给出了芯片引脚到 Leonardo 引脚的映射关系。请注意，并不是所有 ATmega32U4 的引脚都被引出，一些 ADC 输入引脚被用作独立的数字 I/O 引脚。

表3-28：Leonardo ATmega32U4引脚映射1

Leonardo引脚	芯片引脚	功能	Leonardo引脚	芯片引脚	功能
0	20	RXD1/INT2	5	31	OC3A/OC4A
1	21	TXD1/INT3	6	27	T0/OC4D/ADC10
2	19	SDA/INT1	7	1	INT6/AIN0
3	18	OC0B/SCL/INT0	8	28	PCINT4/ADC11
4	25	ICP1/ADC8	9	29	PCINT5/OC1A/OC4B/ADC12

（续）

Leonardo引脚	芯片引脚	功能	Leonardo引脚	芯片引脚	功能
11	12	PCINT7/OC0A/OC1C/RTS	A2	38	ADC5/TMS
12	26	T1/OC4D/ADC9	A3	39	ADC4/TCK
13	32	ICP3/CLK0/OC4A	A4	40	ADC1
A0	36	ADC7/TDI	A5	41	ADC0
A1	37	ADC6/TD0	AREF	42	AREF

3.4 熔丝位

AVR MCU 使用一套熔丝设置各种参数，比如时钟源、时分器（timing divisor）、内存访问锁等。你可以把它想象成一个大的开关面板。本节将讲解 ATmega168/328 的熔丝位。这种 MCU 被用在 Duemilanove、Mini、Nano、Uno 开发板上，并且其 DIP（双列直插式）版本通常被用在 AVR 项目中。相应基本概念也适用于 ATmega1280/2560 与 ATmega32U4。

ATmega168/328 中有 3 B 的熔丝位，分别为低位、高位、扩展位。编程时，熔丝位被设置为 0（例如，它们是低电平有效逻辑）。表 3-29 ~ 表 3-31 列出了 AVR 熔丝位及其功能。

表3-29：低熔丝位

位名称	位编号	描述	取值	位名称	位编号	描述	取值
CKDIV8	7	Clock div by 8	0	CKSEL3	3	Clock source	0
CKOUT	6	Clock output	1	CKSEL2	2	Clock source	0
SUT1	5	Start-up time	1	CKSEL1	1	Clock source	1
SUT0	4	Start-up time	0	CKSEL0	0	Clock source	0

表3-30：高熔丝位

位名称	位编号	描述	取值	位名称	位编号	描述	取值
RSTDISBL	7	External reset disable	1	EESAVE	3	EEPROM preserved	1
DWEN	6	debugWIRE enable	1	BODLEVEL2	2	BOD trigger level	1
SPIEN	5	Enable SPI download	0	BODLEVEL1	1	BOD trigger level	1
WDTON	4	Watchdog timer on	1	BODLEVEL0	0	BOD trigger level	1

表3-31：扩展熔丝位

位名称	位编号	描述	取值	位名称	位编号	描述	取值
-	7	-	1	-	3	-	1
-	6	-	1	BOOTSZ1	2	Boot size	0
-	5	-	1	BOOTSZ0	1	Boot size	0
-	4	-	1	BOOTRST	0	Reset vector	1

AVR 使用时钟输入输出布线逻辑，称为“AVR 时钟控制单元”，如图 3-23 所示。

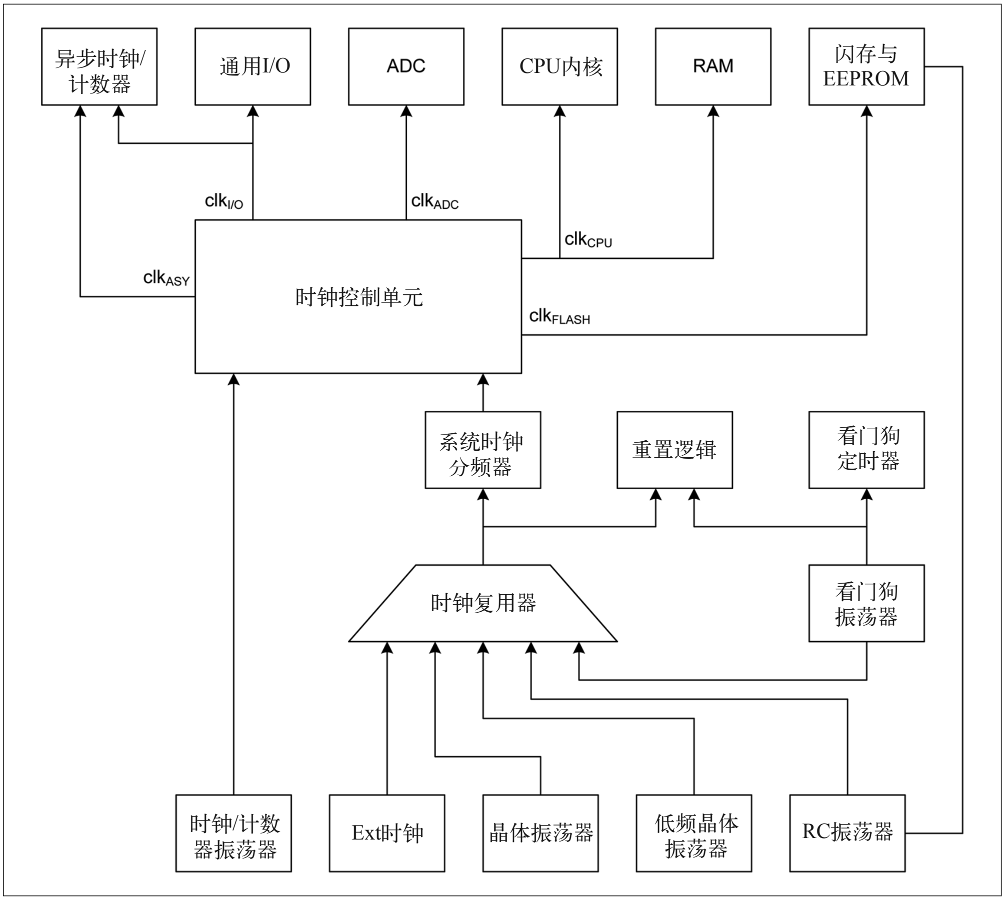


图 3-23：AVR 控制与分配子系统

计时路线 clk 名称指每条路径的定时功能。更多细节请阅读 Atmel 公司关于 ATmega328 的数据手册（http://bit.ly/mc-atmega328）。时钟源多路复用器使用熔丝位 CKISEL（3:0）或者最低有效的 4 位（0.5 B）配置。表 3-32 列出了可能的时钟源选项与 CKSEL 位的位值。

表3-32：使用CKSEL熔丝位选择时钟源

时钟源	CKSEL（3:0）	时钟源	CKSEL（3:0）
低功耗晶体	1111 - 1000	内部校准	0010
最大振荡晶体	0111 - 0110	外部时钟	0000
低频晶体	0101 - 0100	未使用（保留）	0001

请注意，晶振时钟源拥有多个子选项，这取决于所设置的位。在第 10 章，你将会看到这些熔丝位被用于配置定制 PCB 上的一个全新的 AVR。

在基于 Arduino 的设备或系统中，CKSEL 位也许不应该被修改，但可以设置其他熔丝位以便提供有用的功能。在高熔丝位中，WDTON 与 BODLEVEL 熔丝位分别用于启用 AVR 看

门狗定时器以及设置 BOD（掉电检测）的响应级别。看门狗定时器与掉电检测电路会引起 MCU 复位重启。

事实上，看门狗定时器是非常有用的，特别是在某些错误可能会引起更大问题的情况下。只要在超时之前重置看门狗控制寄存器，它通常就不会在主循环中触发中断。如果某些代码不再进行响应，主循环就会停止，并导致看门狗定时器发生超时。avr-libc 为看门狗控制提供了 `wdt_enable()`、`wdt_disable()`、`wdt_reset()` 函数。

当 VCC DC 电源电压降至特定范围之内时，掉电检测就会产生中断。这可以提供足够的时间将关键数据保存到附带的闪存，或更改一些离散的数字输出。

使用 6.5.1 节介绍的编程工具可以对熔丝位进行设置。10.5.3 节中的“为 16 MHz 晶振设置 AVR MCU 熔丝位”部分介绍了如何为一块新的 AVR MCU IC 设置熔丝位，使之以 16 MHz 时钟频率运行。

以上只是从高层介绍熔丝位，关于看门狗定时器与掉电检测级别的更多细节，请阅读 Atmel 公司的相关文档。

3.5 更多信息

第 2 章概述了 AVR 微控制器中常见的内部外围电路。第 4 章将介绍对应于 AVR MCU 功能的 Arduino 开发板引脚。

关于 AVR 微控制器内部外围电路的更多细节，请阅读如下 Atmel 文档（以下文档来自 Atmel 官网 – 支持 – 数据手册）。

- 数据手册：ATmega48PA/ATmega88PA/ATmega168PA/ATmega328P 8 位微控制器，带有 4/8/16/32 kB 的系统内可编程闪存。
- 数据手册：Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V 8 位 Atmel 微控制器，带有 16/32/64 kB 的系统内可编程闪存。
- 数据手册：ATmega16U4/ATmega32U4 8 位微控制器，带有 16/32 kB 的 ISP 闪存与 USB 控制器。

第 4 章

Arduino技术细节

本章介绍 Arduino 开发板的一般物理与电气特性，包括从 Diecimila 到最近的各种开发板，比如 Leonardo、Esplora 与 Micro。主题涵盖引脚描述与当前大部分 Arduino 型号的物理大小，从 Uno 等基本类型到更大的 Mega 开发板和独特的 Esplora，再到 Mini、Micro、Nano 等小型开发板。

4.1 Arduino特性与功能

表 4-1 对大部分常见的 Arduino 开发板进行了比较。如果对比该表与第 1 章中的表就会发现，Arduino 开发板的基本功能都是由其上的微控制器所提供的。然而，由于设计 Arduino 时把 AVR 处理器的某些引脚分配给特定功能，而并非把处理器的所有引脚都进行引出，所以 Arduino 开发板不能使用微控制器提供的所有功能。

> 此处或其他地方提到 Arduino 上的引脚插口时，会使用“引脚”（pin）这一术语。这主要是为了与其他地方碰到的术语保持一致，但从技术上看，它并非完全正确。Arduino 开发板上的连接点是插口，插入这些插口的跳线与扩展板才是真正的引脚。你可以把“引脚”想象成某种连接点，它是 IC 封装上的一根导线，0.1 in（2.54 mm）接口头上的一个位置，或者从扩展 PCB 底部延伸出来的引脚。

表4-1：Arduino硬件特性

开发板名称	处理器	VCC（V）	时钟（MHz）	AIN[a]引脚	DIO[b]引脚	PWM[c]引脚	USB
ArduinoBT	ATmega328	5	16	6	14	6	None
Duemilanove	ATmega168	5	16	6	14	6	Regular
Duemilanove	ATmega328	5	16	6	14	6	Regular
Diecimila	ATmega168	5	16	6	14	6	Regular
Esplora	ATmega32U4	5	16	-	-	-	Micro
Ethernet	ATmega328	5	16	6	14	4	Regular
Fio	ATmega328P	3.3	8	8	14	6	Mini
Leonardo	ATmega32U4	5	16	12	20	7	Micro
LilyPad	ATmega168V	2.7~5.5	8	6	14	6	None
LilyPad	ATmega328V	2.7~5.5	8	6	14	6	None
Mega	ATmega1280	5	16	16	54	15	Regular
Mega ADK	ATmega2560	5	16	16	54	15	Regular
Mega 2560	ATmega2560	5	16	16	54	15	Regular
Micro	ATmega32U4	5	16	12	20	7	Micro
Mini	ATmega328	5	16	8	14	6	None
Mini Pro	ATmega168	3.3	8	6	14	6	None
Mini Pro	ATmega168	5	16	6	14	6	None
Nano	ATmega168	5	16	8	14	6	Mini-B
Nano	ATmega328	5	16	8	14	6	Mini-B
Pro (168)	ATmega168	3.3	8	6	14	6	None
Pro (328)	ATmega328	5	16	6	14	6	None
Uno	ATmega328	5	16	6	14	6	Regular
Yún	ATmega32U4	5	16	12	20	7	Host (A)

a 模拟输入

b 数字 I/O

c 脉宽调制输出（备用 DIO 引脚功能）

4.2 Arduino USB接口

从 Arduino Leonardo 开发板开始（2012 年），ATmega32U4 XMEGA 微控制器就一直被用作主处理器。它拥有一个内置的 USB 接口，不再像早期带有 USB 接口的 Arduino 开发板一样需要使用额外的芯片。Leonardo（2012）、Esplora（2012）、Micro（2012）、Yún（2013）都使用 ATmega32U4 处理器。

带有 USB 的老 Arduino 开发板使用 FTDI 接口芯片（FT232RL）、ATmega8（Uno）或 ATmega16U2（Mega2560 与 Uno R3）。FT232RL 在标准串口（如 RS-232）与 USB 之间进行转换。在 Uno、Uno R3 和 Mega2560 中，外加的小 ATmega 处理器被预编程为 USB 接口。使用 Arduino IDE 创建与加载程序时，这些部件的操作对用户是透明的。

对不带有 USB 接口的开发板进行编程时，必须使用外部适配器。

对于使用 FTDI FT232RL 串口到 USB 接口（serial-to-USB）芯片的 Arduino 开发板，它们的内部结构本质上完全一样，都由一个 DC 稳压电路与两个 IC 组成。图 4-1 为带有一个 FTDI 接口芯片的 Diecimila 与 Duemilanove 的框图。

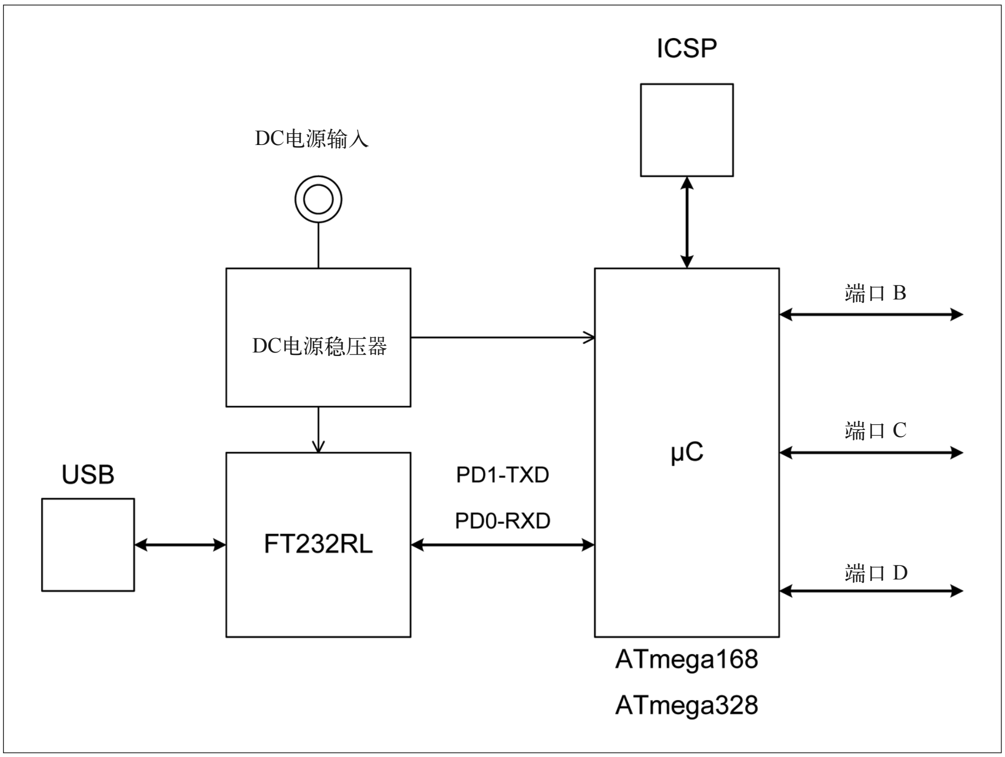


图 4-1：FTDI USB 接口

从 2010 年前后开始，Uno R2 与 Uno SMD 开发板使用 ATmega16U2 代替 FTDI FT232RL，以提供 USB 接口。Uno R3 也把 ATmega16U2 用作 USB 接口。ATmega16U2 内部集成了一个 USB 2.0 接口，它与 ATmega32U4 基本一样，只是拥有更小的内存。图 4-2 显示了 Uno R2 的框图，它使用 ATmega16U2 提供 USB 接口。使用 ATmega8 的 Uno 拥有与 Uno R2 一样的内部功能安排，但它使用一个不同的 MCU 提供 USB 接口。

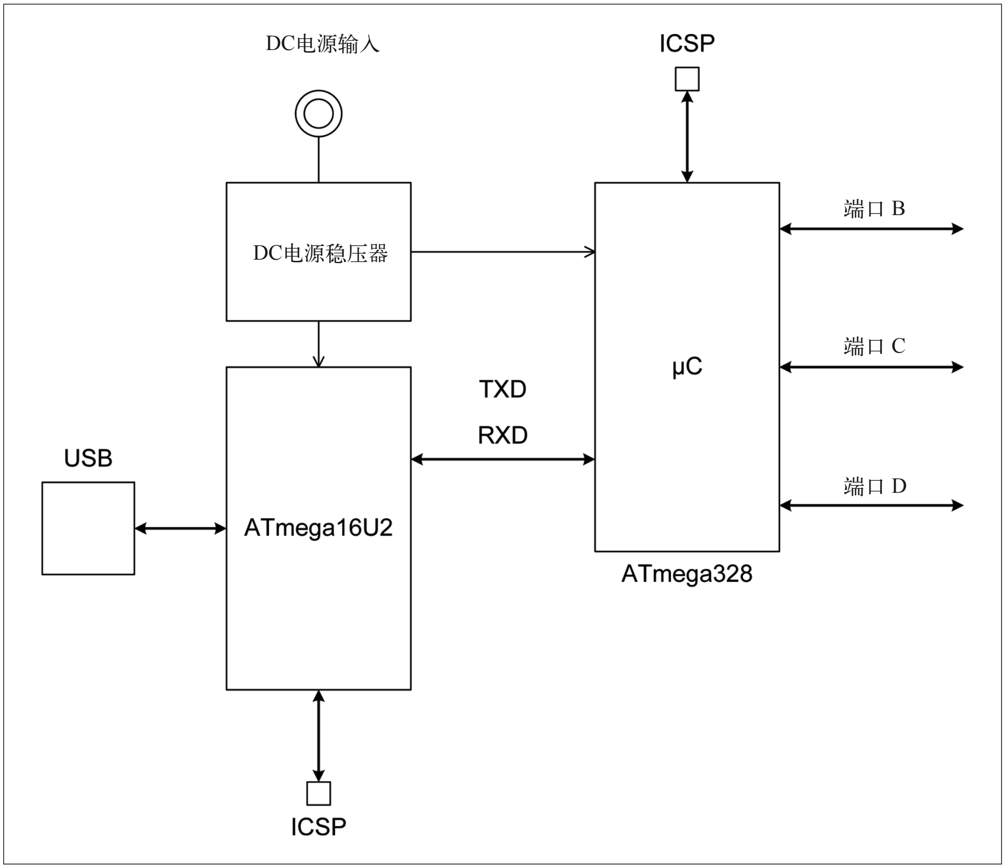


图 4-2：ATmega16U2 USB 接口

4.3 Arduino物理大小

Arduino 开发板的外形和大小多种多样。但一般来说，我们可以把它们分成 4 组，分别为全尺寸或基本开发板、Mega 型开发板、小型开发板，以及特殊用途开发板。

此处给出的开发板大小通常都是近似值，因为不同来源给出的开发板略有不同。你可以从 Arduino.cc 查看 PCB 布局，其中包含每种开发板的大小。如果需要获得开发板的准确规格，更好的办法是亲自动手测量。

为了让大家对 Arduino 大小有个大致的概念，图 4-3 给出了几款常见的 Arduino 开发板实物图。从左下沿顺时针方向，依次是 Duemilanove、Leonardo、带有扩展 I/O 引脚布局的 Mega2560 克隆板（SainSmart 生产）、Arduino 官方 Mega2560，位于中间的是 Arduino Nano。

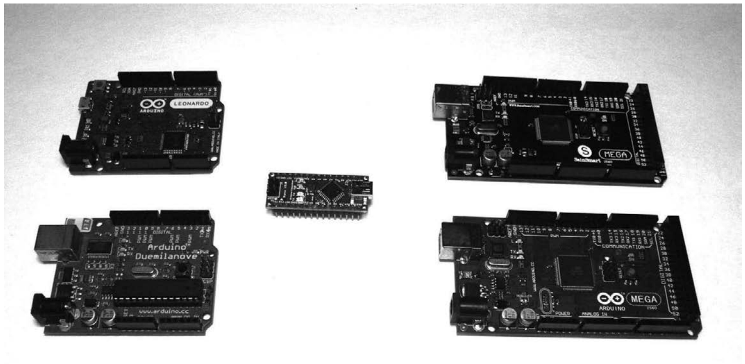


图 4-3：Arduino 与克隆板的比较

4.3.1　全尺寸基本Arduino PCB类型

图 4-4 显示 6 种不同的基本 Arduino 开发板的物理布局，包含从 Diecimila 到 Leonardo 的 6 种开发板，其中还包括 Duemilanove 与 Uno 变形。这里所说的“基本”（Baseline）指经典的 Arduino PCB 布局，它决定着大部分扩展板以及其他附加组件的物理设计。每个 PCB 上的 I/O 功能与其他引脚将在 4.4 节进行讲解。

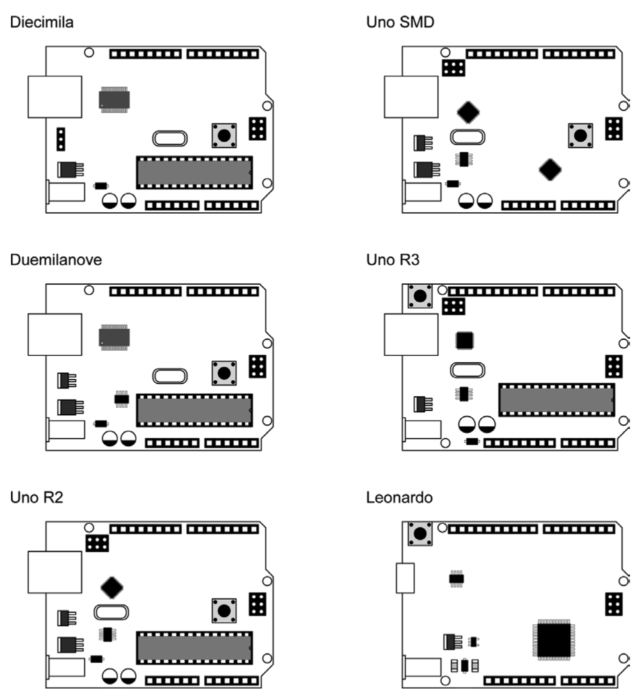


图 4-4：全尺寸基本 Arduino PCB 类型

在 Diecimila、Duemilanove、Uno R2、Uno SMD 开发板上，沿着 PCB 边缘的 I/O 插口头的安排并未发生变化。本书将之称为“基本 Arduino 形态参数”。此外，从 Uno R2 开始，除了早期开发板上已经存在的区块之外，PCB 上又新增了一个 6 个引脚的区块，称为“ICSP（In-Circuit Serial Programming，在线串行编程）接口”，它是为 ATmega16U2 处理器（用作 USB 接口）准备的。Uno SMD 也拥有这个新的 ICSP 接口。

Uno R3 采用了新的扩展 I/O 引脚配置。这是一种向后兼容扩展，意味着老 Arduino 开发板（比如 Duemilanove）的扩展板仍然可以在较新的开发板上正常使用。扩展中只是添加了新的信号引脚插口，而没有添加新的信号，也没有修改基本布局中的任何引脚功能。Leonardo PCB 使用 ATmega32U4 处理器，带有内置的 USB 支持，因此在 Leonardo PCB 上只有一个微控制器和一个 ICSP 端口。它拥有与较早开发板一样的 I/O 引脚布局，但实际使用的微控制器端口是不同的。

所有全尺寸基本 Arduino 开发板拥有一样的物理大小，如图 4-5 所示。不同型号 PCB 上的安装孔的位置略有不同，具体取决于开发板的版本，但整体外形是一致的。

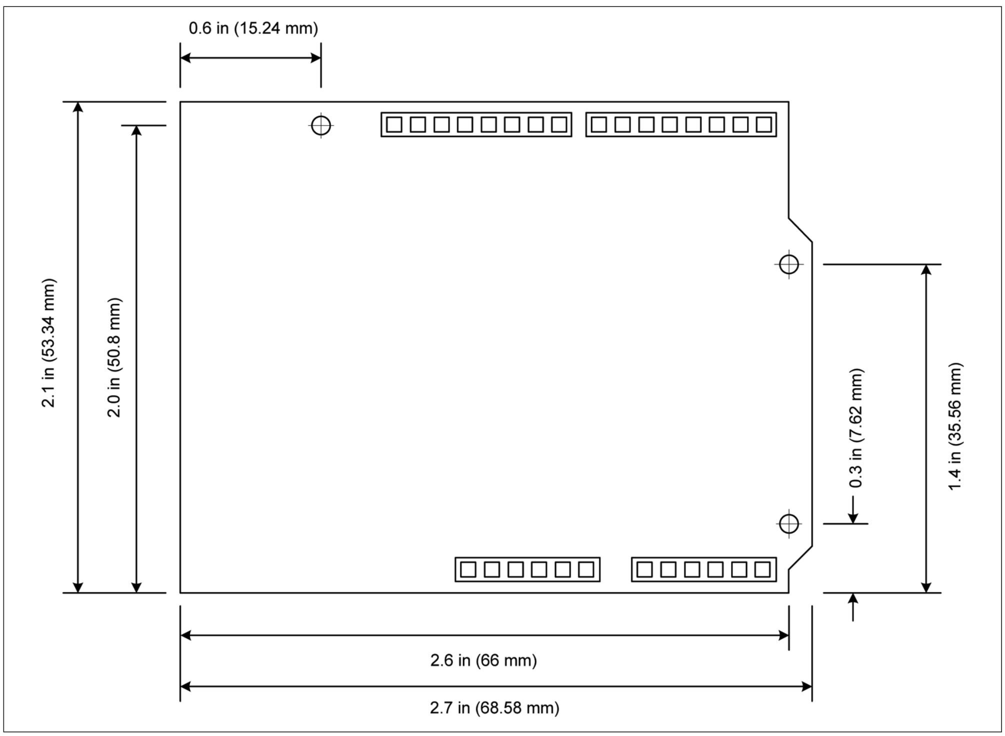


图 4-5：基本板与扩展板的大小

4.3.2 Mega类型的Arduino PCB

Mega 类型的开发板集成了基线引脚，并带有额外引脚，以便适应 ATmega1280 与 ATmega2560 微控制器的扩展能力（这些设备已在 3.2 节进行讲解）。

1. Mega与Mega2560

Mega 与 Mega2560 本质上具有一样的布局，主要区别在于开发板上 AVR 设备的类型。Mega2560 取代 Mega，Arduino.cc 不再生产 Mega，但一些第二方供应商的克隆板仍然可以使用。Mega2560 拥有更大内存，实在没有任何理由再购买 Mega 了。

图 4-6 显示了 Mega 与 Mega2560 开发板的外形大小。请注意，基本扩展板（像 Uno、Leonardo 等）能够在 Mega 开发板之下正常工作。Mega 开发板上的 I/O 引脚安排使得基本数字 I/O 与 A/D 输入 0~5 和基本引脚布局相兼容。

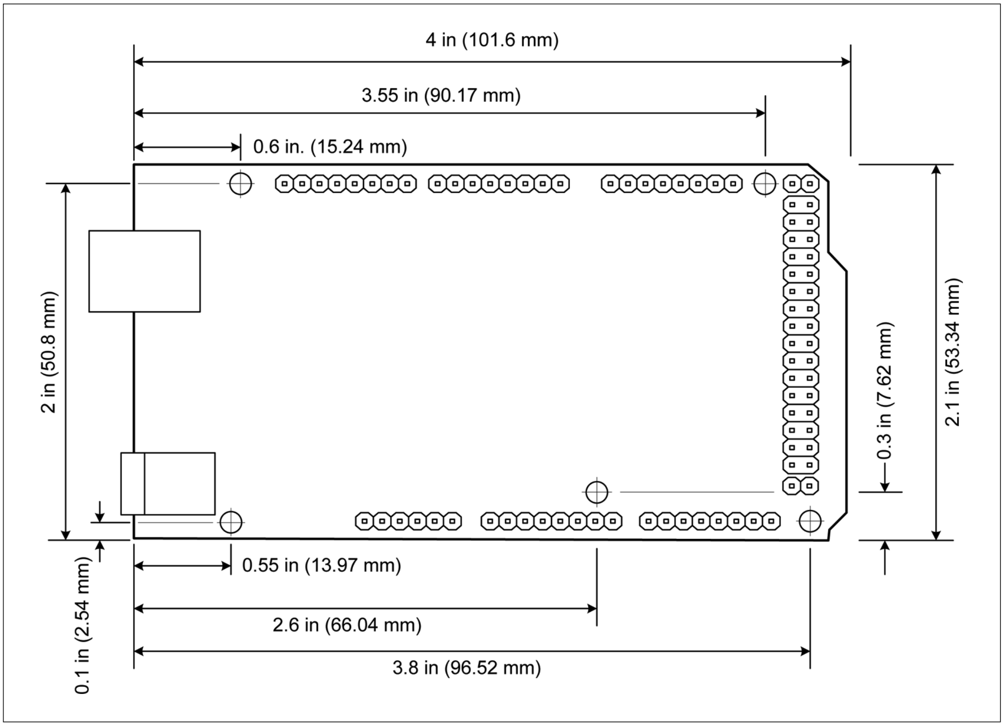


图 4-6：Mega 与 Mega2560

2. Mega ADK

Mega ADK 基于 Mega2560，但拥有一个 USB 主机接口，允许它连接到 Android 手机及类似设备。除了位于 B 型 USB 连接器与 DC 电源接口之间的 USB 连接器之外，从大小来看，它与 Mega2560 完全一样。与 Mega2560 类似，标准的基线型扩展板可以用在 Mega ADK 上。

4.3.3 小型Arduino PCB

全尺寸 Arduino 开发板首次出现在 2005 年前后，到 2007 年，开发板布局最终成型，成为现在你所看到的基本形式与扩展形式。但 Arduino 团队与他们的合作伙伴很快认识到，在某些应用场合下，全尺寸开发板无法很好地工作。为此，他们决定开发小型开发板。

Arduino 小型开发板包括 Mini、Micro、Nano、Fio 几种。在宽度与长度上，这些 PCB 更小，但仍然拥有与全尺寸类型一样的 AVR 处理器。

1. Mini

Mini 开发板配合面包板使用，或者在一些空间受到限制的应用场合之下使用。它不带有 USB 连接器，向微控制器传送可执行代码时，必须使用外部编程接口。Mini 开发板的大小如图 4-7 所示。

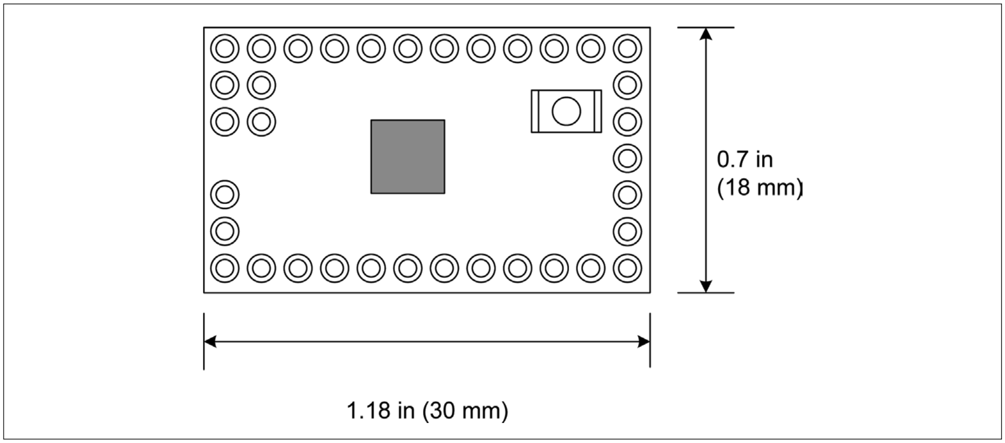


图 4-7：Arduino Mini 大小

2. Pro Mini

Pro Mini 与 Mini 类似，二者拥有相同的引脚布局与外形参数。但与 Mini 不同，Pro Mini 用于永久性安装或半永久性安装，它由 SparkFun 电子设计与制造，其大小如图 4-8 所示。

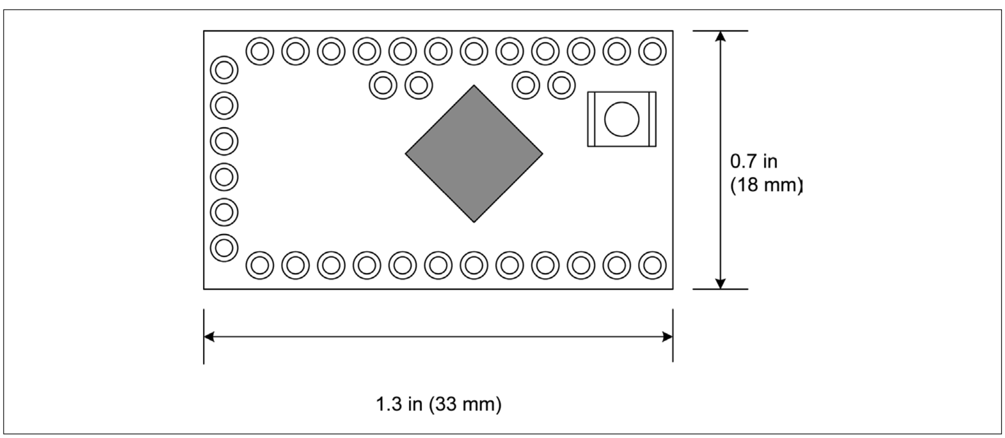


图 4-8：Arduino Pro Mini 大小

3. Nano

与 Mini 类似，Nano 具有较小外形，适合与免焊面包板配合使用，并用作更大 PCB 的插件

模块。它由 Gravitech 设计并制造，其大小如图 4-9 所示。

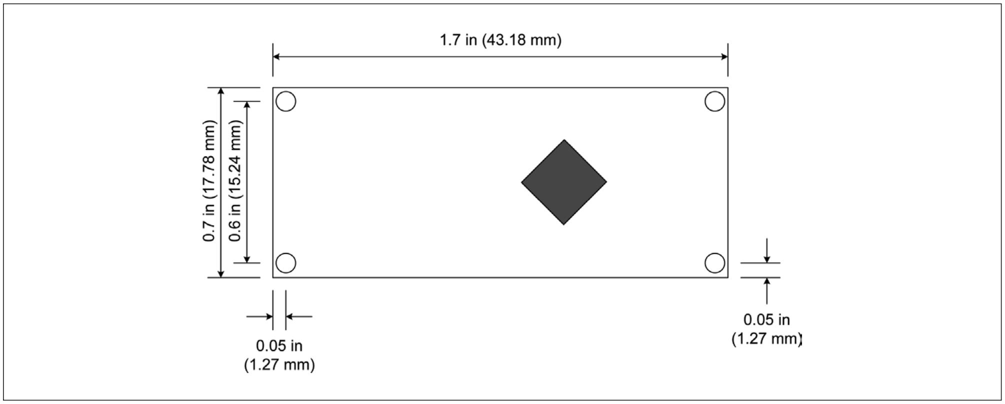


图 4-9：Arduino Nano

4. Fio

Fio 设计用于无线场合或与 XBee 配合使用，它并不像其他 Arduino 开发板一样可以直接进行连接编程。对 Fio 编程时，需要使用串口到 USB 适配器，或者使用 USB 到 XBee（USB-to-XBee）无线适配器。Fio 由 SparkFun 电子公司设计并制造，其大小如图 4-10 所示。

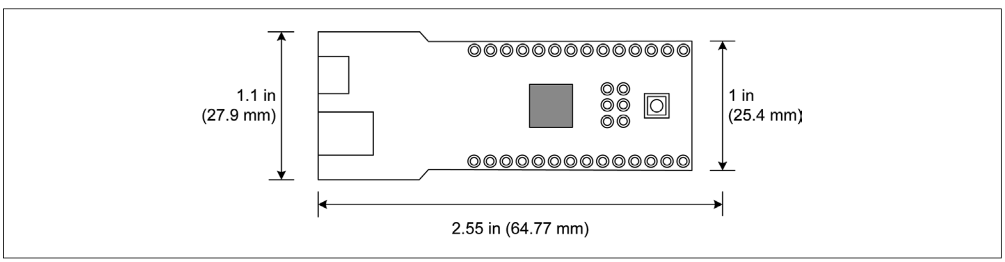


图 4-10：Arduino Fio

5. Micro

Micro 采用 DIP 封装（双列直插式封装）方式，使用 ATmega32U4 处理器，这与 Leonardo 开发板完全相同。与 Nano 类似，Micro 适合与免焊面包板配合使用，并用作插件模块，使用传统的 IC 插口。Micro 由 Arduino 团队与 Adafruit 联合开发，其大小如图 4-11 所示。

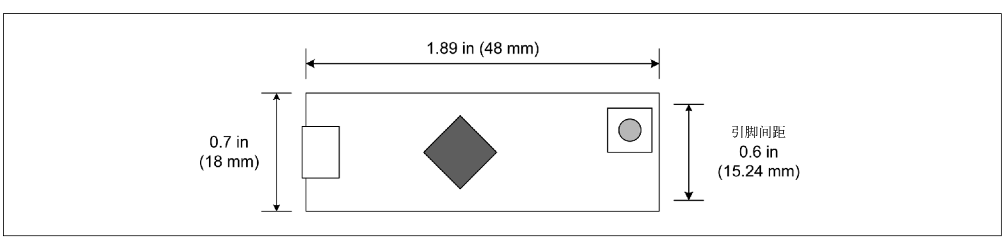


图 4-11：Arduino Micro

4.3.4 特殊用途PCB类型

Arduino 开发板不限于矩形等基本形状。LilyPad 是一个小的圆盘状开发板，连接点沿着圆盘周边设置。它可以缝在织物上，以创建可穿戴应用。从外形上看，Esplora 像一个传统的游戏控制器，但它是一款 Arduino 设备，用户可以对它进行编程，创建各种应用，而不仅仅用来玩游戏。

1. LilyPad

LilyPad 及其各种变型专为可穿戴应用而设计，它是一款圆盘形开发板，直径约为 2 in（50 mm），如图 4-12 所示。

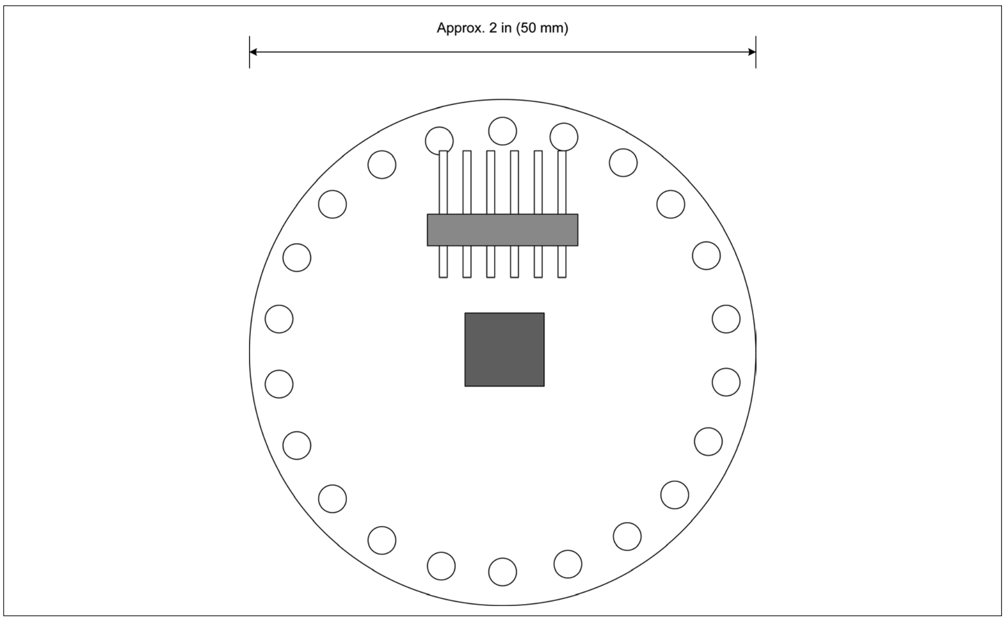


图 4-12：Arduino LilyPad

2. Esplora

Esplora 配备有 4 个按钮、一个开关型操纵杆，以及一个 USB 连接器。Esplora 拥有 4 个可用的安装孔，方便将其固定到一个底盘或面板上。Esplora PCB 大小如图 4-13 所示。

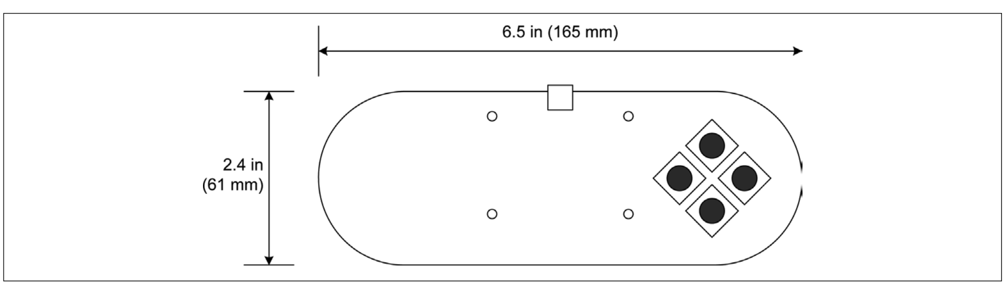


图 4-13：Arduino Esplora

4.4 Arduino引脚布局

为 Arduino 创建"背板"（shield board）时，约定遵从此处介绍的常见基线引脚布局方式。在 2007~2012 年生产的标准基本 Arduino 开发板上，可以看到这样的结构布局。使用更新的"扩展"引脚布局的开发板（Uno R3 与 Leonardo），包括 Mega 开发板，也支持基线连接，但要通过扩展靠着 PCB 边缘的终端行添加新功能。

4.4.1 Arduino基线引脚布局

现在所见的 Arduino 基线引脚布局最早出现在 Diecimila 型号上。经过数年的发展，它成为事实上的标准，许多背板都遵循这一标准。采用基线引脚布局的 Arduino 开发板如表 4-2 所示。

表4-2：基线布局Arduino开发板

型号名	年份	微控制器
Diecimila	2007	ATmega168
Duemilanove	2008	ATmega168/ATmega328
Uno (R2 and SMD)	2010	ATmega328P

图 4-14 显示了一块全尺寸基线 Arduino 板的引脚，其中涵盖的型号有 Diecimila、Duemilanove、Uno R2、Uno SMD。图 4-14 中的灰色框给出了 ATmega168/328 器件的芯片引脚编号与命名。

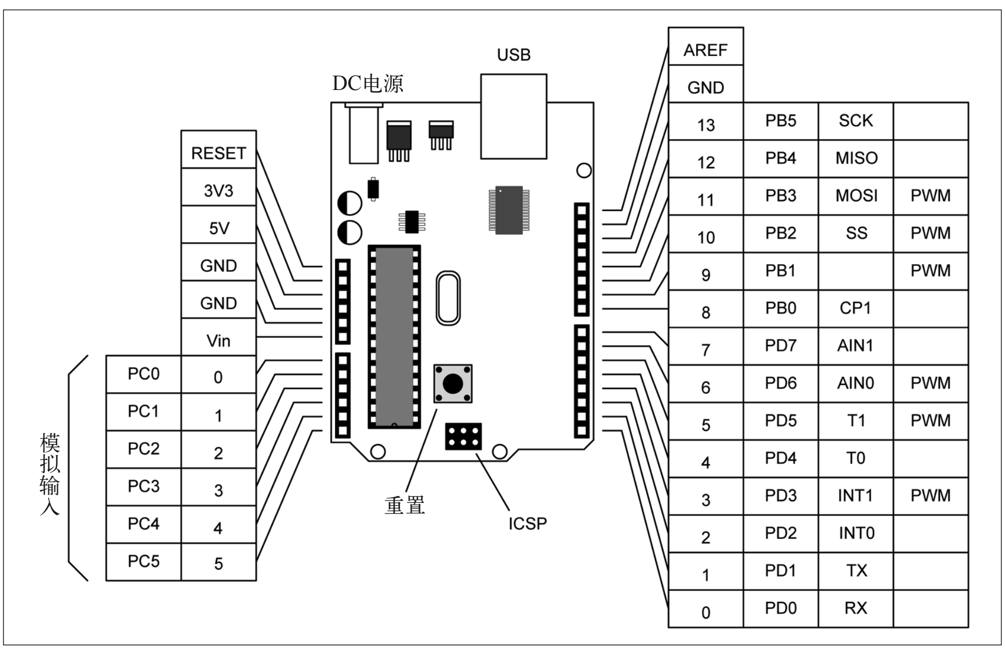


图 4-14：标准基线 Arduino 板的引脚布局

Arduino 常见的基线 I/O 与电源引脚布局由 14 个独立数字 I/O 引脚、一个模拟参考（analog reference）、3 个接地引脚、6 个模拟输入引脚、3.3 V 与 5 V 引脚，以及一个复位线组成。如图 4-14 所示，沿着 PCB 边缘，这些引脚被设计为两个 8 位连接器与两个 6 位连接器。

从编程角度看，Diecimila、Duemilanove、Uno R2、Uno SMD PCB 上的每个接口引脚都拥有一个预定义名，用于在软件中对其进行标注。这些名字被以标签的形式印制到 Arduino PCB 上。表 4-3 列出了基线板或 Arduino R2 板（采用 ATmega168 或 ATmega328 MCU）的引脚分配。请参看 Arduino Ethernet（表 4-3）与 Uno SMD 板的引脚分配。

表4-3：Arduino ATmega168/328引脚分配

数字引脚（Dn）	模拟引脚（An）	AVR引脚	AVR端口	AVR功能	AVR PWM
0		2	PD0	RxD	
1		3	PD1	TxD	
2		4	PD2	INT0	
3		5	PD3	INT1, OC2B	Yes
4		6	PD4	T0, XCK	
5		11	PD5	T1	Yes
6		12	PD6	AIN0	Yes
7		13	PD7	AIN1	
8		14	PB0	CLK0, ICP1	
9		15	PB1	OC1A	Yes
10		16	PB2	OC1B, SS	Yes
11		17	PB3	OC2A, MOSI	Yes
12		18	PB4	MISO	
13		19	PB5	SCK	
14	0	23	PC0		
15	1	24	PC1		
16	2	25	PC2		
17	3	26	PC3		
18	4	27	PC4	SDA	
19	5	28	PC5	SCL	

4.4.2 扩展基线引脚布局

从 Uno R3 开始，Arduino PCB 上新增了 4 个引脚，其中 2 个位于复位按钮附近，用于为 I2C（SCL 与 SDA 线）提供附加连接。另外 2 个引脚紧挨着开发板另一侧的复位连接，一个用作 IOREF（额定 I/O 电压为 3.3 V 或 5 V，具体取决于开发板类型），另一个尚未连接使用。表 4-4 列出了扩展基线布局开发板。

表4-4：扩展布局Arduino开发板

开发板名称	年份	微控制器
Uno R3	2010	ATmega328
Ethernet	2011	ATmega328
Leonardo	2012	ATmega32U4

1. Uno R3

类似于 Uno R2 与 Uno SMD，Uno R3 采用单独的微控制器处理 USB 通信。Arduino Ethernet 不带有内置 USB。图 4-15 是 Uno R3 与 Uno SMD 开发板的框图。

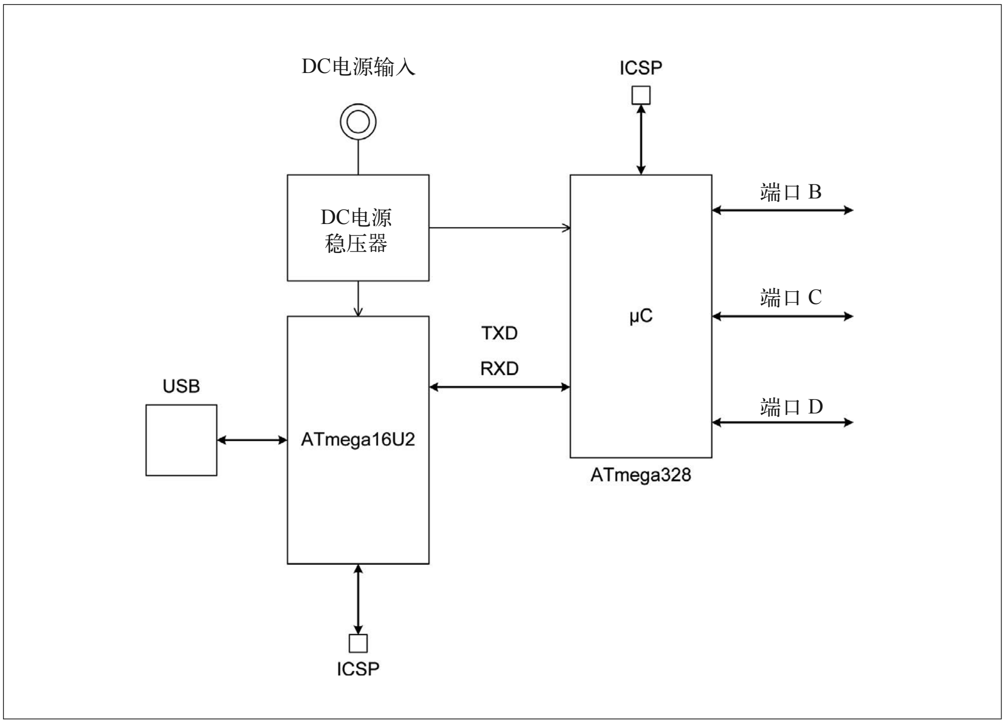


图 4-15：ATmega16U2 USB 接口

Uno R3 的引脚功能如图 4-16 所示。

带有 ATmega328 MCU 的扩展基线 Arduino 开发板（R3）拥有与表 4-3 相同的引脚分配，但拥有 ADC4 与 ADC5 两个额外引脚（A4 与 A5）。下一小节将介绍 Leonardo 引脚功能定义。

2. Ethernet

Ethernet 背离了 Uno R3 中常见的 Arduino 约定，它带有一个 100 MB 的以太网接口和一个 RJ45 插口，不带有 USB 接口。MCU 是 ATmega328 的表面贴装版本，拥有不同于 ATmega328 的引脚功能与编号。采用 WIZnet W5100 芯片作为以太网接口。图 4-17 是 Ethernet 开发板的框图。

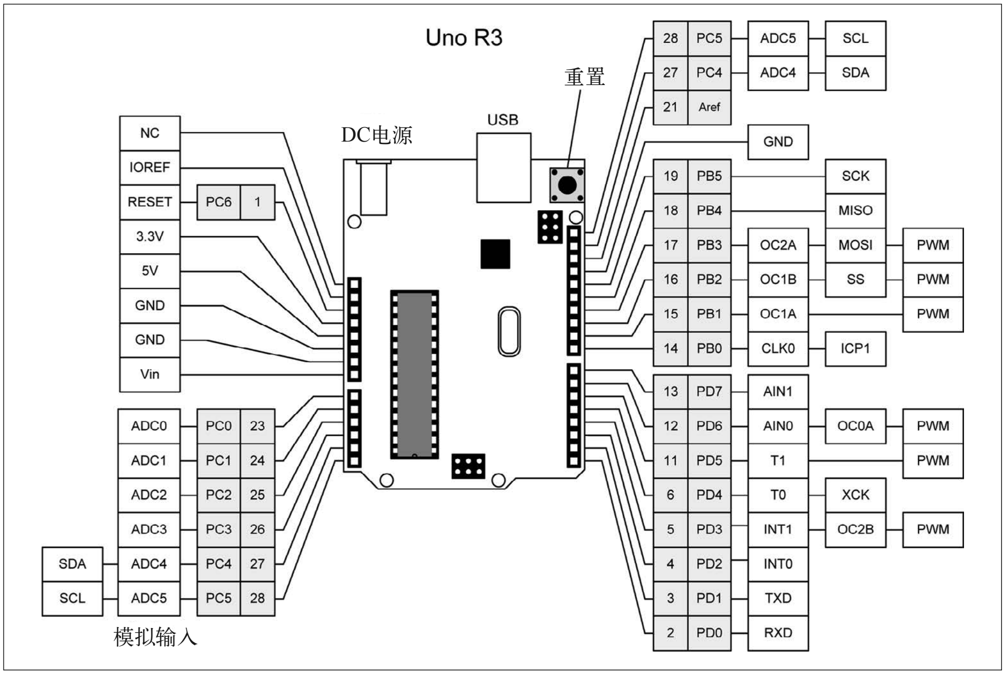


图 4-16：Uno R3 引脚功能

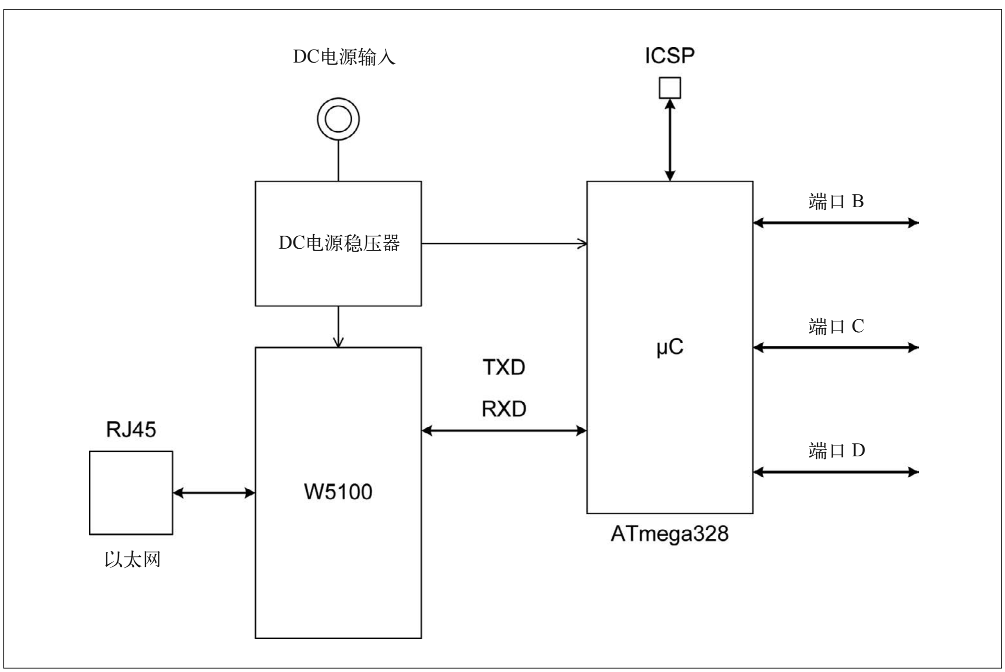


图 4-17：Arduino Ethernet 框图

类似于 SparkFun 或 Adafruit FTDI 类型的设备，对带有适配器的 Ethernet 进行编程时，要用到 FTDI 类型的接口。该接口从 PCB 边缘的直角六引脚头引出，紧挨着 microSD 卡槽。图 4-18 显示了 Ethernet 开发板的引脚。

该产品已经被 Arduino.cc 淘汰，但你仍然可以从多种渠道购买。通过使用以太网扩展板可以实现以太网连接（有关扩展板的详细内容请参考第 8 章）。

表 4-5 列出了 Arduino Ethernet 的引脚分配情况。请注意，第 10/11/12/13 号引脚专为以太网接口保留，不要把它们用于普通用途。

表4-5：Arduino Ethernet引脚分配

数字引脚（Dn）	模拟引脚（An）	AVR引脚	AVR端口	AVR功能	AVR PWM
0		30	PD0	RxD	
1		31	PD1	TxD	
2		32	PD2	INT0	
3		1	PD3	INT1, OC2B	Yes
4		2	PD4	T0, XCK	
5		9	PD5	T1, OC0B	Yes
6		10	PD6	AIN0, OC0A	Yes
7		11	PD7	AIN1	
8		12	PB0	CLK0, ICP1	
9		13	PB1	OC1A	Yes
10		14	PB2	OC1B, SS	Yes
11		15	PB3	OC2A, MOSI	Yes
12		16	PB4	MISO	
13		17	PB5	SCK	
14	0	23	PC0		
15	1	24	PC1		
16	2	25	PC2		
17	3	26	PC3		
18	4	27	PC4	SDA	
19	5	28	PC5	SCL	

3. Leonardo

Leonardo 采用 ATmega32U4 处理器，含有一个内置的 USB 接口与增强功能。这简化了 PCB 布局，如图 4-19 所示。此外，请注意，Leonardo 使用一个 mini-USB 连接器，取代旧 Arduino 开发板上使用的全尺寸 B 型连接器。这是迫切需要做出的改变，它允许 Leonardo 与可能干扰 B 型 USB 连接器的扩展板配合工作。

Uno R3 与 Leonardo 使用相同的 PCB 引脚布局，但它们之间微控制器的一些功能是不同的。在 Arduino IDE 中，通过使用特定于某种类型开发板的一系列定义，将各种功能映射到特定端口。

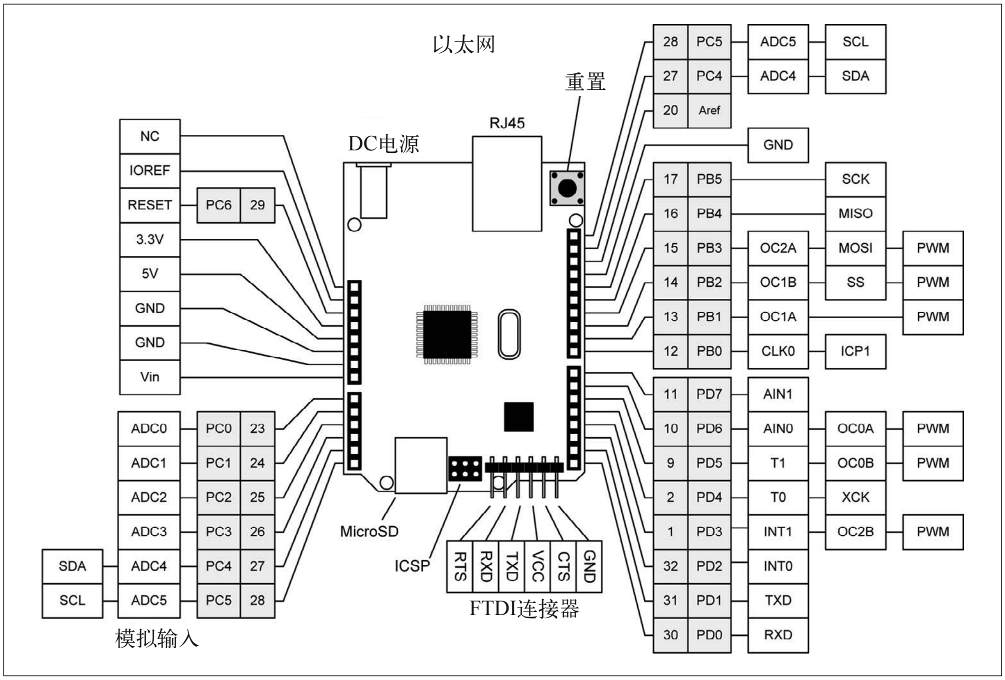


图 4-18：Arduino Ethernet 开发板引脚功能

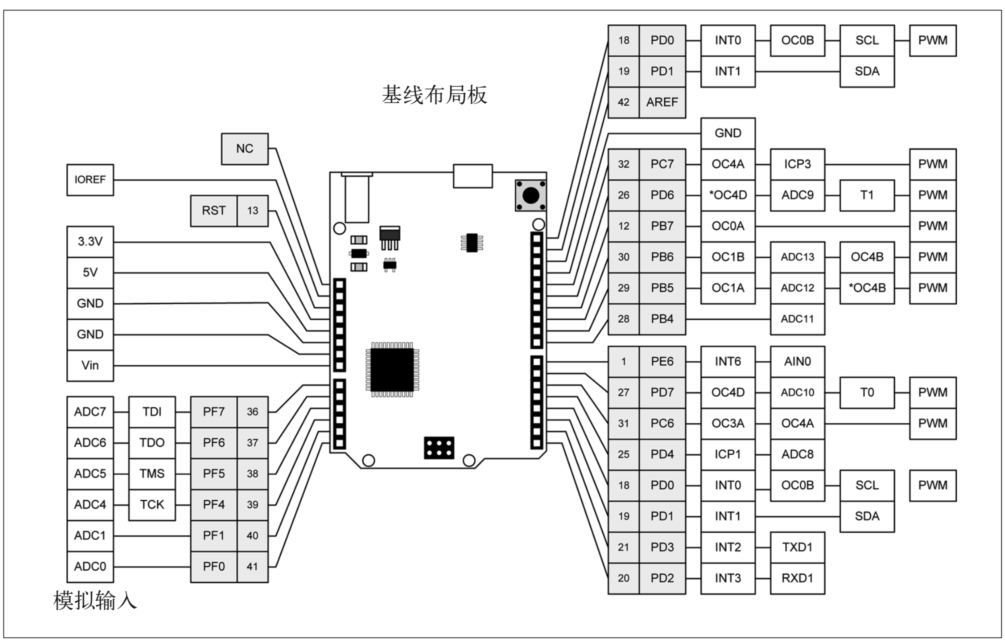


图 4-19：Arduino Leonardo 开发板引脚功能

表 4-6 列出扩展 Arduino 或 Arduino R3（采用 ATmega32U4）的引脚分配情况。

表4-6：Arduino ATmega32U4引脚分配

数字引脚（Dn）	模拟引脚（An）	AVR引脚	AVR端口	AVR功能	AVR PWM
0		20	PD2	INT3, RxD1	
1		21	PD3	INT2, TxD1	
2		19	PD1	INT1, SDA	
3		18	PD0	INT0, OC0B, SCL	Yes
4		25	PD4	ICP1, ADC8	
5		31	PC6	OC3A, OC4A	Yes
6		27	PD7	OC4D, ADC10, T0	Yes
7		1	PE6	INT6, AIN0	
8		28	PB4	ADC11	
9		29	PB5	OC1A, ADC12, *OC4B	Yes
10		30	PB6	OC1B, ADC13, OC4B	Yes
11		12	PB7	OC0A	Yes
12		26	PD6	*OC4D, ADC9, T1	Yes
13		32	PC7	OC4A, ICP3	Yes
14	0	36	PF7	TDI	
15	1	37	PF6	TDO	
16	2	38	PF5	TMS	
17	3	39	PF4	TCK	
18	4	40	PF1		
19	5	41	PF0		

4.4.3　Mega引脚布局

Mega 系统开发板（采用 ATmega1280 与 ATmega2560 处理器）也集成了标准的引脚类型，但是拥有额外的引脚，以适应更大处理器的扩展 I/O 性能。Mega 引脚布局如图 4-20 所示。采用这种布局的开发板如表 4-7 所示。大部分常见扩展板都能与 Mega 系列开发板协同工作。

图 4-20 中，为了简明起见，并未标注 PCINT 引脚。此外，还请注意早期版本中不存在 Mega2560 的 R3 版本包含的引脚，但它们并不会影响基线布局类型的扩展板。

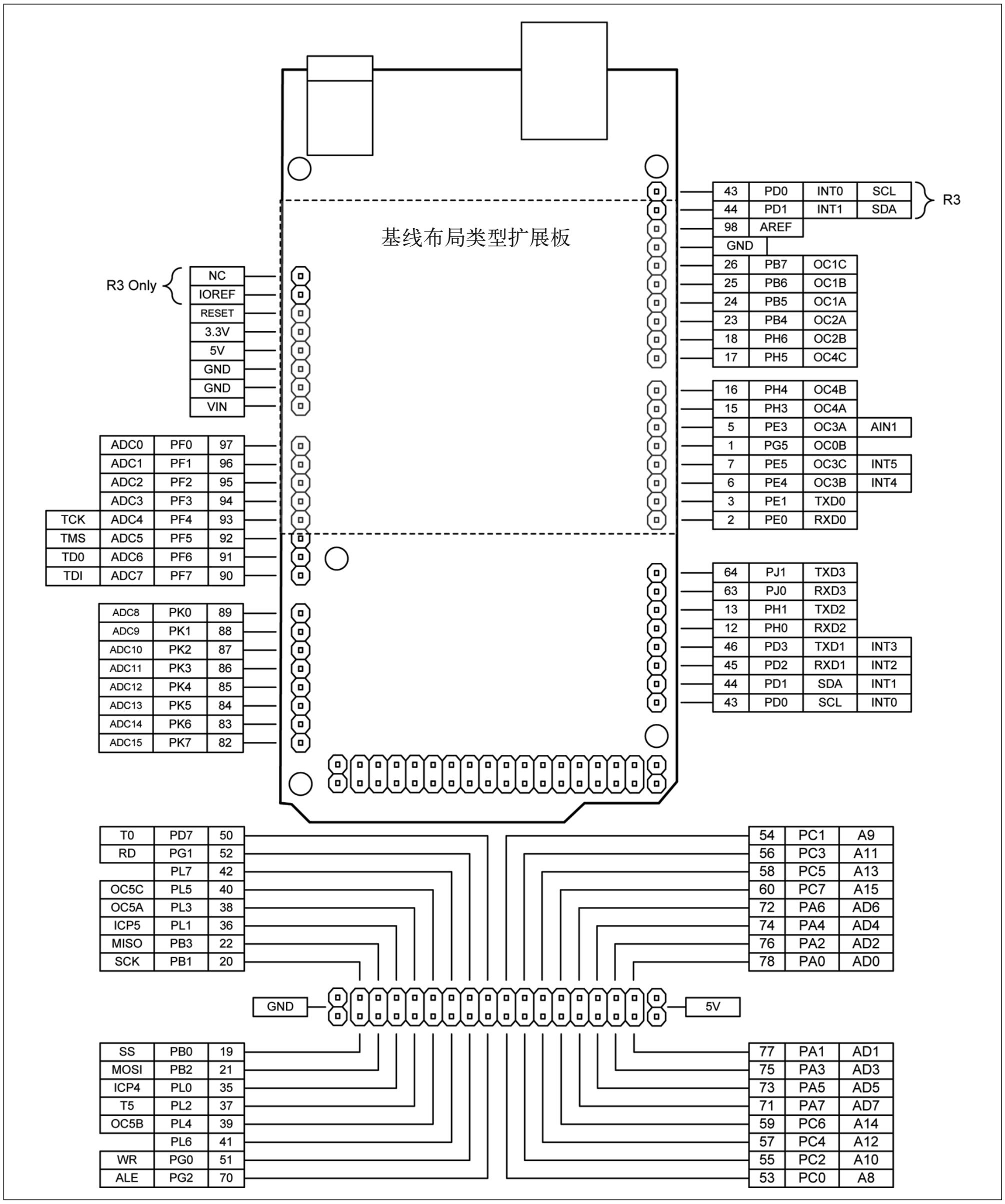


图 4-20：Arduino Mega 系列开发板的引脚功能

表4-7：Mega系列开发板引脚布局

开发板名称	年份	微控制器
Mega	2009	ATmega1280
Mega2560	2010	ATmega2560
Mega ADK	2011	ATmega2560

4.4.4 非标准布局

从非标准引脚布局来看（非标准意味着与传统 Arduino 扩展板物理不兼容），最激进的当数 LilyPad。它拥有圆形外形，使用焊盘（solder pads）进行连接。小型板 Nano、Mini、Mini Pro、Micro 有引脚焊接到板子底面，适合与免焊面包板配合使用，或者用作更大 PCB 上的一个组件。Fio 使用带有间距的焊盘，与标准排针相兼容。Esplora 拥有类似游戏控制器的外形。任何采用非标准布局方式的开发板都不能直接与标准扩展板配合使用。这类开发板在表 4-8 中列出，它们的引脚功能如图 4-21~ 图 4-27 所示。

表4-8：非标准引脚布局开发板

开发板名称	年份	微控制器
LilyPad	2007	ATmega168V/ATmega328V
Nano	2008	ATmega328/ATmega168
Mini	2008	ATmega168
Pro Mini	2008	ATmega328
Fio	2010	ATmega328P
Esplora	2012	ATmega32U4
Micro	2012	ATmega32U4

1. LilyPad

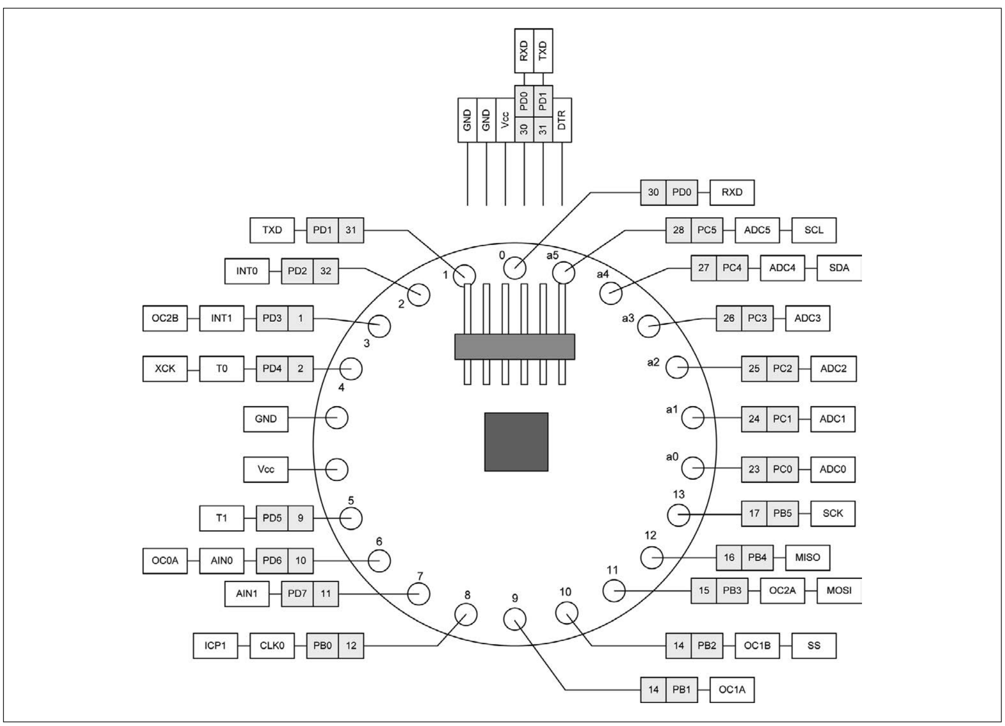


图 4-21：Arduino LilyPad 的引脚功能

2. Nano

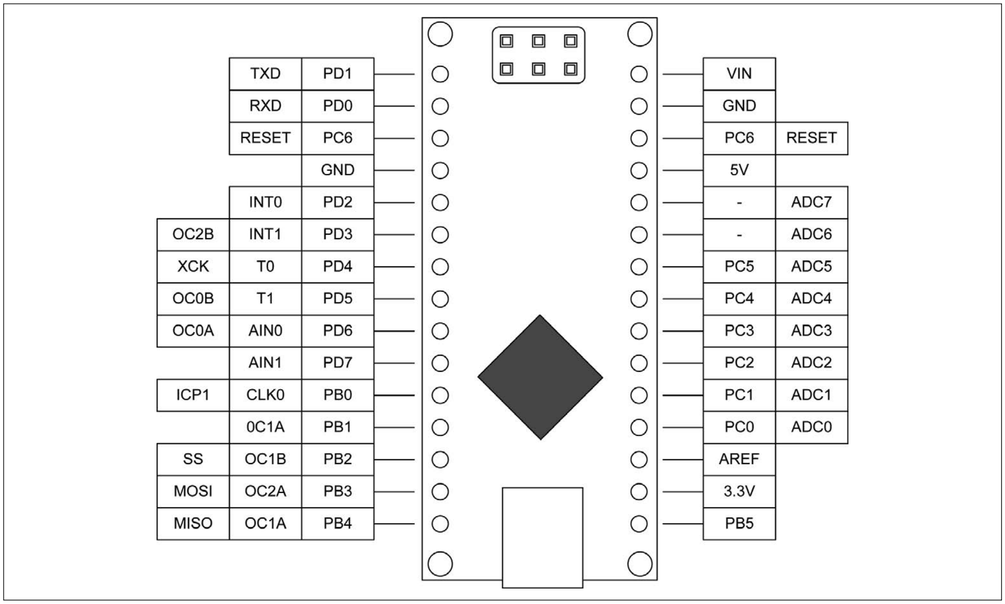


图 4-22：Nano 引脚功能

3. Mini

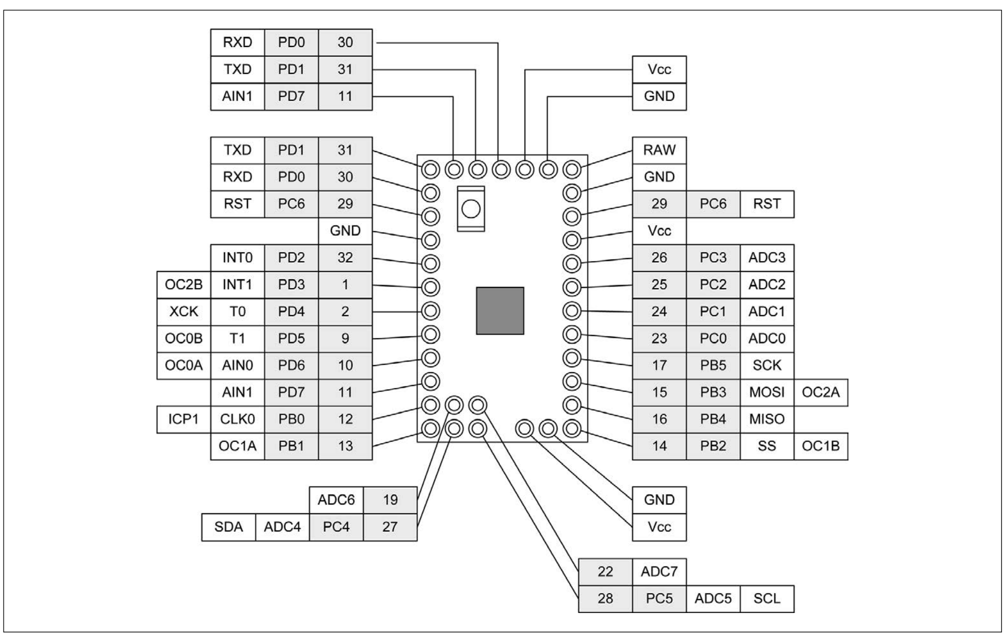


图 4-23：Mini 引脚功能

4. Pro Mini

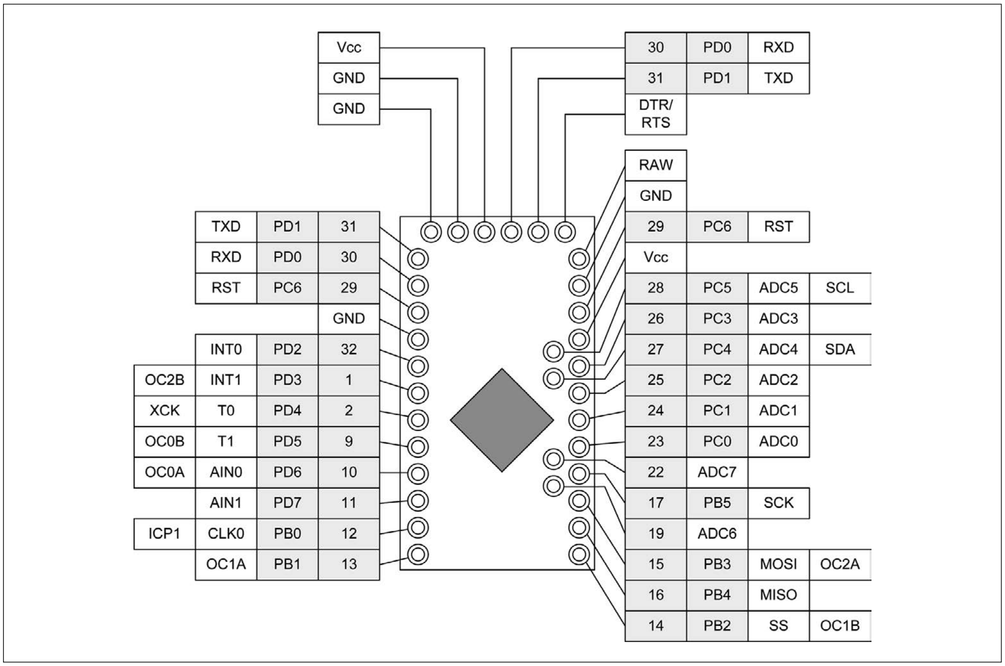


图 4-24：Pro Mini 引脚功能

5. Fio

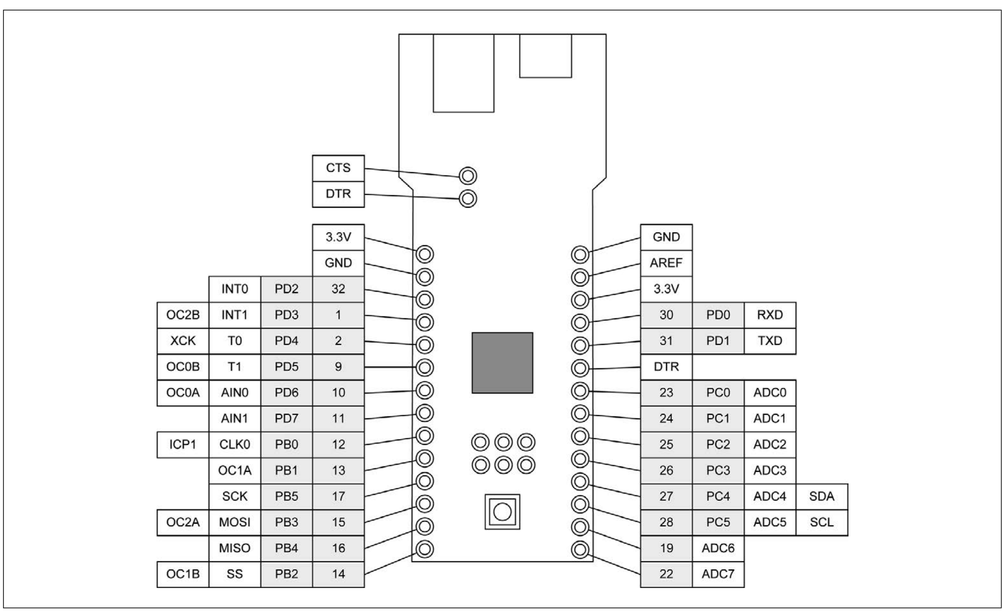


图 4-25：Fio 引脚功能

6. Esplora

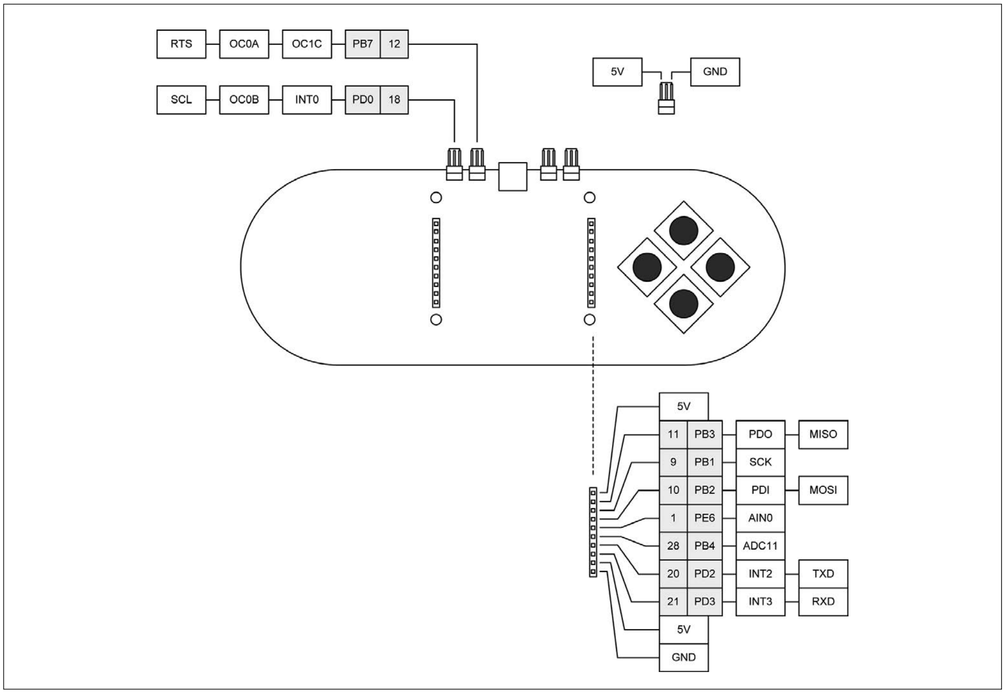


图 4-26：Esplora 引脚功能

7. Micro

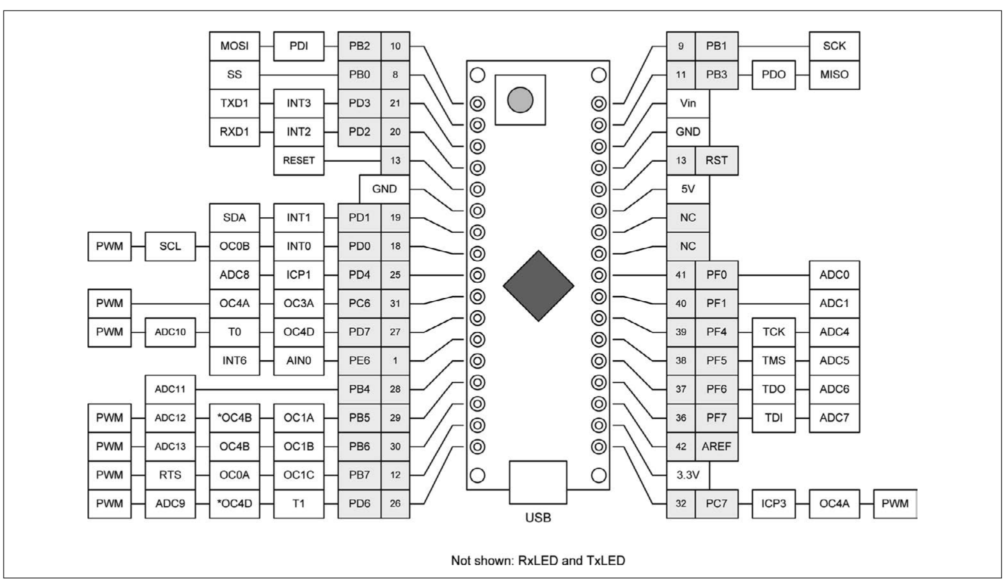


图 4-27：Micro 引脚功能

4.5 更多信息

Atmel 官方站点（http://www.atmel.com/avr）提供了关于 AVR 微控制器的一系列数据手册、示例代码和其他资源。请注意，这些内容都是关于 AVR 的，而不是关于 Arduino 的。

要想获得有关各种 Arduino 开发板的更多信息，可以访问 Arduino 官方站点（http://www.arduino.cc）。

第 5 章

对Arduino与AVR微控制器编程

本章将简要介绍一些与编程有关的工具、概念与技术，借助它们，你可以创建、编译、汇编代码，并把它们加载到 Arduino。此处只对这些主题做大致讲解，其实每个主题背后都有更深的内容，但实在无法只使用一个章节就把它们全部讲完。本章的目标是向你提供足够信息，帮助你获得使用微处理器的经验。通过本章，你将发现一些从未注意到的有关 Arduino 环境的内容。

> 本章不会讲解 C 与 C++ 语言，请阅读附录 D 中推荐的图书并学习相关内容。本章旨在帮助各位理解如何将程序转换为二进制代码，以便在 AVR MCU（Arduino 开发板或其他采用 AVR MCU 的开发板）中运行，并了解这一过程中有哪些因素参与。

本章先简要介绍交叉编译技术，它允许开发者在一个计算机系统上使用编译器与其他工具创建可执行程序，所创建的程序可以被传送到另外一台具有完全不同架构的计算机中执行。这正是 Arduino 集成开发环境要做的事。第 6 章将学习有关 Arduino IDE 使用的开发工具与技术的更多内容，此处只关注 Arduino IDE 能做什么，以及如何高效地使用它。

接着讲解 BootLoader，重点是 AVR 微控制器。BootLoader 是用于 Arduino 开发板上的 AVR 系列器件的主要技术特征，理解其工作原理将有助于减少学习中的挫败感，也为将来安装你自己的 BootLoader 打好基础（有时我们的确需要这样做）。第 6 章将学习如何安装新 BootLoader（也可以是你自己设计的）。

之后介绍 Arduino IDE，包括如何将 IDE 安装到各种计算机中，以及如何按照自己的使用习惯对其进行配置。我们也会简单了解一下 IDE 背后都发生了什么，以及它是如何从早期工具发展而来的。并通过一个简单的示例程序向各位展示创建一个 Arduino 程序（称为 sketch）的要点。

最后，本章将带领各位一起浏览 Arduino 源代码。这一过程中，你可以得到自己要检查的代码副本。对于所提供的库、IDE 及其组件，拥有可用的源代码能够帮助回答诸如“为什么它会以这种方式发生”的问题，也能帮助你决定是否真想尝试使用一个替代方案为 Arduino 进行编程。

> 请记住，本书讲解的重点是 Arduino 硬件及其相关模块、传感器、组件，书中给出的示例代码都是软件中最关键的部分，它们并不是完整的可运行代码。关于示例与项目的完整软件代码，请前往 GitHub（https://www.github.com/ardnut）进行下载。

5.1 微控制器交叉编译

与大多数单板微控制器系统类似，我们需要先使用另一种系统（Windows、Linux 或 Mac）为 Arduino 及其 AVR MCU 开发程序，然后将程序传送或上载到 Arduino 开发板上的 AVR 微控制器。开发系统被称为“*开发主机*”，Arduino 或者其他基于 MCU 的设备被称为“*目标*”。这种开发过程已经存在了很长一段时间，当我们需要为自身不具备编译代码能力的目标机器编写程序时，就会使用它。

在一个不同类型的系统上为一种处理器编制软件的技术称为“交叉编译”，它是用来为微控制器目标创建编译好的软件的唯一方式。AVR 这类较小的微控制器（或者任何一款小型的 8 位设备）没有足够的资源编译与链接 C 或 C++ 程序。我们一般会使用性能更强大且带有更多资源（比如更快的 CPU、更大硬盘、更多内存）的机器对程序进行编译与链接，然后将制作好的程序传送到目标机上运行。

有时，主机系统中甚至会有目标机的模拟器，它允许开发者在模拟环境中加载与测试程序。模拟器可能无法 100% 地对目标 MCU 及其实际环境进行模拟，但它仍然是一个非常有用的工具。将软件上载到实际的 MCU 之前，可以借助模拟器测试软件的基本功能。Linux 操作系统下有一些可用的 AVR 模拟器，比如 simavr 与 GNU AVR 模拟器（附录 E 列出了 Arduino 与 AVR 软件工具以及相关链接，感兴趣的读者可以参考）。

目标代码、映像和源代码

我们谈论编译、链接、加载软件时，可能会经常见到“目标代码”“映像”“可执行的”“源代码”这些术语。它们都非常古老，可以追溯到使用穿孔卡、存储磁鼓、磁带、体型大到可以占满多个房间的计算机的时代。

如你所料，源代码是编译器或汇编器的输入，它是人类可读的，由开发人员使用某种文本编辑器创建。对于 Arduino，源代码通常都是使用 C 或 C++ 语言，并使用 Arduino IDE、其他编辑器或 IDE 编写的。

编译器或汇编器输出的是目标代码，它是特定于某种 CPU 的机器代码。换言之，它由 CPU 操作码的二进制值和关联数据（以字面值形式存在，也是二进制形式）组成。目标代码通常称为“机器语言”。现代编译器与汇编器对源代码进行汇编或编译处理后，都会生成 *.o 与 *.obj 文件，其中包含目标代码。

目标代码可能无法立即运行。如果程序由两个或更多模块组成，或者程序需要从外部库导入目标代码，那么编译器或汇编器的输出将包含一些占位符，这些占位符指代外部代码。

有时候，外部代码可能以源代码形式存在（比如大部分 Arduino 代码），它们将与你的代码一起进行编译。外部代码也有可能已经被编译成包含一个或多个独立代码模块的目标库。通常，预编译的目标库带有 *.a 扩展名。

链接器（linker）工具用于填充“空白”，将各部分代码连接成一个可执行映像。就 Arduino 程序而言，这通常是指，将运行时支持以 `main()` 函数的形式包含进来，还包括基本的 AVR 运行时库函数，如基本的输入 / 输出函数，以及其他低级操作。

最后得到的可执行映像包含程序运行所需的一切，它们都在一个包中。转换成十六进制字符串组成的 ASCII 文件之后，程序被上传到 AVR MCU；接着转换回二进制，并存储在板载闪存中等待执行。

5.2 BootLoader

有几种不同的方法可以将一段程序上传到现代微控制器，但其中最简单的是让 MCU 帮助处理这一过程。可以通过使用一小段 BootLoader 实现。有关固件的讨论，请阅读下面的补充栏目“固件起源”部分。

AVR 系列微控制器在板载闪存中为 BootLoader 提供了预留空间。一旦 MCU 被配置为使用 BootLoader，那么当 AVR MCU 加电或重启时，会先到这块特殊内存空间所在的地址处查找指令。只要 BootLoader 未被上传的程序重写，那么在开 / 关电源之间，它将一直驻留在内存中。

通常，保留 BootLoader 的位置包含启用内部开关或熔丝，这通过一个特殊的编程设备实现，该设备通过 ICSP 或 JTAG 接口与 MCU 进行通信。MCU 启动时会检查熔丝位配置（非易失性配置位），并据此决定如何组织闪存，以及是否为某种 BootLoader 或其他启动代码预留空间。（关于 AVR 熔丝位更多细节，请参考 3.4 节）。

AVR 设备的一个主要特征是，它们有能力通过 BootLoader 与串口，将程序代码加载到自己的内部闪存。就 ATmega32U4（第 2 章和第 3 章介绍过）来说，使用 USB 接口可以把程序代码上传到 MCU，而无需使用特殊的编程设备或辅助 MCU。

Arduino 开发板（包括官方开发板与软件兼容开发板）已经将自身 BootLoader 装载到 MCU。ArduinoBootLoader 实现了一种特殊的协议，该协议让 BootLoader 识别出 Arduino IDE，并允许用户把程序数据从开发主机传送到目标开发板。有关 ArduinoBootLoader 的细节，将在 6.5.7 节中讲解。对于如何把 BootLoader 替换为你自己选择的 BootLoader，相关技术将在 6.5.8 节进行讨论。

固件起源

“固件”这一术语是历史遗留的产物。那时，嵌入式计算机本身带有预载程序，并且这些程序是无法再次进行修改的。有时候，微控制器或微处理器会装配软件，这些软件已经内置到硅芯片之中。开发者通过二进制机器码指出芯片生产商，二进制机器码会被并入制造实际硅芯片的过程。这被称为“掩模编程”（mask programming），其优点是程序可靠、成本低（批量购买时），但它不是最实用的软件开发办法。它通常需要大量的开发支持设备，还有特定版本的目标微控制器。在某些情况下，那意味着要使用在线仿真器（ICE），它与目标处理器在功能上是等效的。目标处理器拥有独立的逻辑 IC，包含在金属盒中，通过带状电缆插入插口，从而将处理器安装到电路板。如果代码工作正常，且通过了全面测试，那就可以在芯片上较为可靠地实现代码。也就是说，我们假设 ICE 能够精确模拟实际的微处理器或微控制器。然而，使用一些较早版本的工具时，情况并非总是如此。

在其他情况下，固件可能被加载到一个芯片上。该芯片使用一种特殊的一次性编程设备，类似于编程时使用的只读存储器存储芯片。事实上，这些编程设备一般都能用来操纵 ROM 与 MCU 设备。程序以内存位（Memory bits）的形式存储在芯片中，它们只能被设置一次表示 0 或 1 的值。这被称为 OTP，或者“一次性可编程”器件。对于一次性可编程器件，一旦进行编程就会一直保持不变，无法再次修改。

如果预期的芯片产量不高，不足以支撑掩模方法的开发设置成本，那么 OTP 会是一个合理的替代方案。在小批量生产环境中，使用名为 gang programmers 设备的情况并不少见。它拥有多行专用插口，可以同时对多个 OTP 处理器进行编程。

然而，OTP 仍然需要花费大量努力，使用专用工具进行预先开发与测试。虽然把带有缺陷的固件加载到 OTP 部件中并不像下单生产 10 000 个带有缺陷的掩模部件那样会造成严重的经济损失，但细微代码错误导致 OTP 部件报废的情形仍然令人痛心。

随着技术的进步，新型设备陆续出现。它们带有 UV-EPROM（通常可以在采用 DIP 或双列直插式封装的部件中看到，带有透明的玻璃窗口，允许通过 UV 光线照射芯片以重置内存位）与 EEPROM。这两种部件都需要使用特殊的工具，以擦除只读存储器中的内容并加载新数据。加载了错误代码只意味着需要再次执行“擦除 - 加载 - 测试”这一过程，而并不需要把设备丢入垃圾桶。

现在，大部分最新型的微控制器都使用闪存，但 OTP 部件仍然可用。闪存的一个主要优点是，它是可读写的。换言之，处理器运行时，用户既可以向里面写数据，也可以从里面读数据。另一个主要优点是，它是非易失性的。也就是说，在开 / 关电源时，其中的数据会一直保存。“固件”这一个概念起源于非易失性的只读存储器（需要专用工具才能修改其中数据），已经成为明日黄花，但人们现在仍然将加载到微控制器闪存中的“东西”称为“固件”。

5.3 Arduino IDE环境

写作本书之时，Arduino IDE 最新版本是 1.6.4。不论在哪种平台，Arduino IDE 都呈现一样的用户界面。图 5-1 显示的是 Arduino IDE 启动后出现的初始界面。

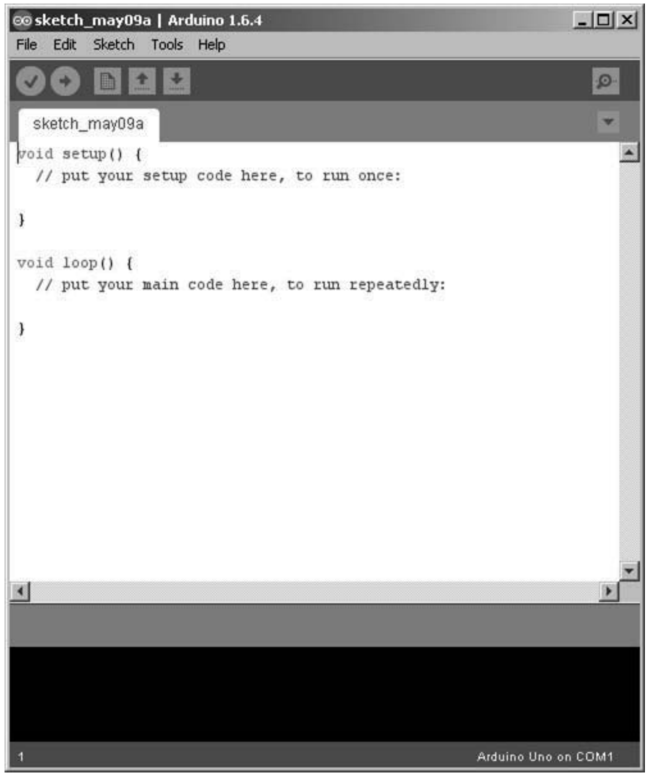


图 5-1：Arduino IDE 主界面

最新版本的 IDE 加入了针对一些菜单的修复与更新，添加了一个用于管理开发板与库的命令行界面，并且解决了一些代码高亮显示的问题。此外，还添加了对新 Gemma 开发板的支持，也提供了对非 Arduino（即非官方）开发板的支持。它允许你输入一个 URL 地址，让 IDE 从生产商网站获取开发板的相关信息，并把它集成到 IDE 的开发板管理器中。

如果你有一个较旧的 Arduino 开发板（比如 Diecimila、Duemilanove、Uno），那么不使用最新版本的 IDE 也不会有什么大问题。大多数拥有较旧型号 Arduino 开发板并且使用较旧版本 IDE 的用户甚至没有注意到，他们根本没有使用最新版本的 IDE。如果你也处于这种境地，除非打算使用更新型的 Arduino 开发板（比如 Yún、Zero、Gemma），否则就目前而言，完全没有必要安装最新版本的 IDE。

5.3.1 安装Arduino IDE

安装 Arduino IDE 与库的难易程度各不相同，有时极其容易，有时又非常复杂，这主要取决于你使用哪种平台，以及想花多少工夫进行安装。对于使用 Linux 操作系统的用户，最简单的方法是使用包管理器下载安装包，并让它帮忙安装所需的各种支持包。

然而，这样做的不足之处在于，从包存储库下载的软件可能不是最新版本。Arduino 官方站点（http://arduino.cc）总是提供最新版本的软件，但收集所有必需的包并让它们能够在 Linux 上正常运行可能需要很多手动操作步骤。相比而言，在 Windows 与 Mac OS X 系统下安装 Arduino IDE 则要容易得多，接下来会讲到。

安装 Arduino IDE 之后，最好先花几分钟浏览一下系统中都安装了什么。在 Arduino 安装目录中，你将看到大量示例的源代码、常用库的源代码与示例、各种 BootLoader 的 hex（二进制可加载映像）文件，以及以 HTML 页面形式展现的文档。你可以使用 Web 浏览器进行查看，也可以使用 IDE 中的 Help 菜单打开。

可以从 Arduino.cc 官方站点（http://www.arduino.cc/en/Main/Software）下载 Arduino IDE，它针对不同操作系统推出多种版本。根据你所用的操作系统，选择相应版本进行下载。

1. Windows

在 Windows 系统中安装 Arduino IDE 非常简单，程序安装包（arduino-1.6.4-windows.exe）会自动处理所有安装细节。如果你的系统中已经安装了较早版本的 IDE，那么安装新版本之前，安装程序会先把旧版本从系统中移除（但不会移除之前编写的程序）。Arduino 可执行文件与库文件会被安装到 Program Files\Arduino 文件夹之下，你也可以在其中找到示例代码。Arduino 目录在你的主目录下创建，所有用户创建的程序与用户提供的库都位于其中。

2. Linux

大多数 Linux 发行版以预先打包好的形式提供 Arduino IDE，如 deb、rpm、ymp 或其他包格式。这是快速获取 Arduino IDE 与库，以及在 Linux 系统中正确进行安装的最佳方法。如前所述，这种方法的不足之处在于，各种 Linux 发行版的软件包维护者可能没有把 Arduino.cc 官网中最新版本的 IDE 放入发行包。比如，针对 Kubuntu，Arduino 最新可用版本为 1.0.4。

使用发行包是在 Linux 系统中安装 Arduino IDE 最简单的方法。你可以直接从 Arduino.cc 下载各种组件，并进行手工安装。对于某些发行版本，这可能是唯一的选择。在 Arduino 站点（http://bit.ly/apg-linux），你可以阅读对 Linux 发行版支持的有关内容。

在类 Ubuntu 系统（Ubuntu、Kubuntu 或其他基于 Ubuntu 的发行版）中，你可以使用 apt-get 或类似工具安装软件包。OpenSuse 系统使用 yast 工具安装 IDE 软件包，其他系统可能使用 rpm 或其他一些包管理器。安装完成后，各种示例、硬件、库文件、工具被安装到 /usr/share/arduino 中。在我的 Kubuntu 和 Xubuntu 系统中，我的程序与定制库被安装到默认位置，即 home 目录下名为 sketchbook 的子目录中。

3. Mac OS X

Arduino IDE 与相关库的 Mac OS X 版本以 ZIP 文件形式提供，名为 arduino-1.6.4-macosx.zip。根据 Arduino 站点的说明，它适用于 Mac OS X 10.7（Lion）或更新版本。对 arduino-1.6.4-macosx.zip 进行解压缩之后，将 Arduino.app 复制到系统的合适位置（Applications 文件夹是最常用的位置之一）。

关于在 Mac OS X 系统中安装 Arduino IDE 的更多信息，请参考 Arduino.cc 站点的相关内容（http://www.arduino.cc/en/Guide/MacOSX）。

5.3.2 配置Arduino IDE

可以通过“首选项”（Preferences）对话框对 IDE 进行定制，使其更好地满足自身需要。在 Mac 系统下，你可以在主菜单中依次选择 File-Preferences 打开首选项对话框。在首选项对话框中，你可以设置项目文件夹的位置，指定使用外部编辑器（如果你不喜欢 IDE 自带的编辑器），以及调整 IDE 的各种行为。图 5-2 为 Linux 系统中较旧版本的 IDE 中的首选项对话框。

在图 5-2 中，你可以看到首选项对话框对应的文件为 /home/jmh/.arduino/preferences.txt，其中包含更多未在对话框中显示的设置。但请注意，不要在 IDE 处于运行状态时对其进行编辑。

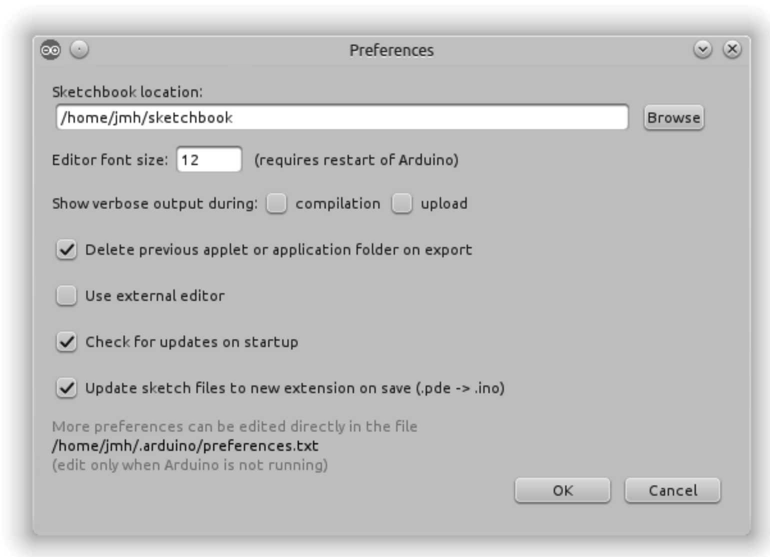


图 5-2：旧版本 IDE 的首选项对话框

图 5-3 显示的是 Arduino IDE 1.6.4 的首选项对话框。与旧版本相比，它拥有更多选项可供设置，但它仍然只为首选项文件提供了图形化界面。

首选项文件是一个 ASCII KVP（键值对）数据文件，其中包含有关编辑器、应用程序初始显示形状（Arduino IDE 窗口大小）、串口参数（波特率、数据大小等）、用于浏览 HTML 帮助文档的浏览器等。

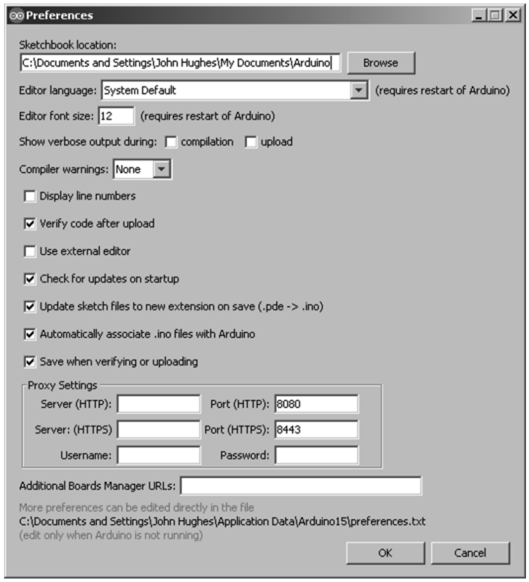


图 5-3：新版本 IDE 的首选项对话框

首选项文件也包含与最后编辑的程序有关的信息，以及 IDE 窗口的最后大小。由于 IDE 在运行时会动态修改首选项文件，所以只能在 IDE 未运行时进行手工编辑。

对于旧版本的 IDE，直接编辑首选项文件可能是修改某些参数的唯一方法，比如缩进量（默认为 2 个空格，我喜欢设置为 4）、编辑器中使用的字体类型与大小（默认字体大小为 12 的 Monospaced 字体，但我一般把字体大小设置为 11，这样在编辑窗口中就能看到更多行）、选项卡扩展。新版本的 IDE 在首选项对话框中提供了更多设置项，但你可能仍然想直接在 preferences.txt 文件中设置某些项目。

5.4　使用Arduino IDE进行交叉编译

让 Arduino 从众多带有 AVR 的 PCB 中脱颖而出的是，Arudino.cc 所开发的 Arduino IDE、Arduino 固件、运行时代码、软件库，当然还有你（开发者）编写的程序。Arduino IDE 是为 AVR 芯片创建与加载软件的一种快速且简单的方式。因为在对代码进行编译、链接并将其传送到 AVR 目标时，Arduino IDE 为我们隐藏了大量底层细节。

> 我们通常在 Arduino IDE 中使用 C++ 语言编程，当然你也可以使用 C 语言，AVR-GCC 工具链都支持它们。因此，同时提及两种语言时，会使用 C/C++ 这种表达方式；而单独讨论其中一种语言时，则会使用“C 或 C++”这种表达方式。

单个源代码文件在 IDE 中分别以“标签”的形式显示。通过简单地选择编辑窗口顶部相应标签，你可以从一个文件切换到另一个文件。编译源代码时，IDE 将遍历每个标签，并为它们分别生成目标文件。

Arduino IDE 使用 avr-gcc 编译器和相关工具——称为“工具链”（tool-chain）——为 AVR 设备编译可执行的二进制代码。Arduino.cc 提供的各种库包含大量有用的函数，比如时间延迟、串行数据输出以及其他功能（详见第 7 章）。

有时，Arduinoese 这一术语用来指 Arduino 使用它自己独特的 C/C++“方言”。这是不正确的。Arduino IDE 中使用的语言的确是 C 或 C++，只是对于使用 C++ 源代码可以做什么会有一些限制。回想一下，Arduino IDE 其实只是对 AVR-GCC 工具链（有关 AVR 工具链的内容请参见 6.2 节）的一层“包装”（wrapper）。

需要特别指出的是，avr-libc 并不支持 C++ 中的 `new` 和 `delete` 操作符，但 Arduino 团队提供了它们。你可以在 Linux 系统下的 /usr/share/arduino/hardware/arduino/cores/arduino/new.h 文件中，或者在 Windows 系统下的 C:\Program Files\Arduino\hardware\arduino\avr\cores\arduino 中，或者在 Mac 下的 /Applications/Arduino.app/Contents/Java/hardware/arduino/avr/cores/arduino 中找到 Arduino 定义。

使用 new 操作符实例化类对象

我们通常认为，在嵌入式系统中动态分配内存是一个“坏主意”。一般而言，动态分配的内存来自 SRAM，大部分 MCU（包括 AVR）没有很多 SRAM 支持这种做法。avr-libc 支持 `malloc()` 操作，但并不支持 `new` 操作。然而 Arduino 团队却认为支持 `new` 操作是一个不错的主意，他们是这么想的，也是这么做的。

请注意，在 AVR C++ 代码中实例化一个类对象时，并不是在 SRAM 空间中创建对象。8 位 AVR MCU 是哈佛架构处理器，它们不从 SRAM 中执行代码，而只从闪存空间中执行代码。其中的诀窍是编译时在闪存中创建对象，`new` 操作返回指向对象的指针。

如示例 5-1 所示，在包含全局变量的源文件中，可以在 1.0.5 或以后版本的 IDE 中使用 `new` 操作符。

示例 5-1　使用 new 操作符实例化全局对象

```
// global_objs.cpp
#include "Arduino.h"
#include <LiquidCrystal.h>
#include <DDS.h>

LiquidCrystal *lcd = new LiquidCrystal(LCD_RS, LCD_E, LCD_D4, LCD_D5,
                                       LCD_D6, LCD_D7);
DDS *ddsdev = new DDS(DDS_OSC, DDS_DATA, DDS_FQ_UP, DDS_W_CLK, DDS_RESET);
```

如果 lcd 与 ddsdev 在相应的包含文件中被声明为导出，如示例 5-2 所示，那么能从包含全局 include 文件的其他任何一个源模块中访问它们。

示例 5-2　将全局对象声明为 extern

```
// global_objs.h
#include <LiquidCrystal.h>
#include <DDS.h>

extern LiquidCrystal *lcd;
extern DDS *ddsdev;
```

或者，你也可以先将全局变量文件中的对象实例化，再将其指派给相应指针（此处可以使用同样的包含文件），如示例 5-3 所示。

示例 5-3　将对象指派给全局指针

```
// global_objs.cpp
#include "Arduino.h"
#include <LiquidCrystal.h>
#include <DDS.h>

LiquidCrystal lcdobj(LCD_RS, LCD_E, LCD_D4, LCD_D5, LCD_D6, LCD_D7);
DDS ddsobj(DDS_OSC, DDS_DATA, DDS_FQ_UP, DDS_W_CLK, DDS_RESET);

LiquidCrystal *lcd = &lcdobj;
DDS *ddsdev = &ddsobj;
```

有意思的是，这些声明的可执行对象拥有不同的编译大小，具体大小取决于所用的方法。new 操作产生一个 662 B 的可执行对象，比指针分配方法大。虽然不是太大，但如果你只有 32 kB 的闪存可用，就不得不慎重考虑。

这些代码片段取自信号发生器源代码，你将在第 11 章看到它们。

除此之外，C++ 模板也是不可用的，异常处理也不被支持。而类（包括构造函数与析构函数）是被支持的。使用纯 C 代码几乎没有限制，甚至可以使用 malloc() 与 free()，但使用时要注意一些事项，更多细节请阅读 avr-libc 用户手册（http://bit.ly/avr-libc-malloc）。请记住，从技术上来说，在嵌入式 MCU 中使用动态内存分配是不可靠的，因为这可能导致一些问题，使代码变得臃肿、笨拙。古话说得好："你能做并不代表着你应该这样做。"要在一个内存受到限制的微控制器上使用动态内存分配，必须有令人信服的理由。

对于 Arduino IDE 使用的语言，引起困惑的源头可能是其中预定义的宏以及各种函数，Arduino 环境使用它们来简化对 AVR MCU 各种 I/O 函数的访问。Arduino 程序可能有一个古怪的名称和并不常见的文件扩展名（.ino），看上去像是精简版的 C，但这是有意为之。如果你认为 Arduino 的目标受众是那些没有任何编程经验的人，这也是完全合理的。但如果你想使用，那么所有 C 与大部分 C++ 的复杂性仍然存在且可用。第 6 章将介绍如何使用基本的 AVR-GCC 工具链组件、文本编辑器和 makefile 文件为 Arduino（或任何 AVR MCU）创建可执行代码。

微控制器不是微处理器

重要的是，要牢记微控制器并不是一般用途的 CPU。典型的微控制器是一个设施齐全的设备，带有有限的程序内存与非常少的 RAM，但内建有大量 I/O 功能。微控制器通常用来执行一些特殊任务，比如记录键盘上的击键动作、控制发动机、监测温度，或者为动态艺术项目提供智能控制，或者同时执行这些任务。ATmega2560 是个例外，它有能力访问外部存储器。就文字处理器或复杂的视频游戏而言，AVR 微控制器其实不是一个合适的平台。话虽如此，聪明的"玩家"总是使用简洁、高效的代码，设法让 AVR 微控制器去做更有趣、更复杂的事情。

5.4.1 Arduino可执行映像

通常，Arduino 可执行软件（此前称为可执行映像）由 3 个主要部分组成：开发者（你）编写的程序、用于访问 AVR 各种函数的库、运行时代码（包含 `main()` 函数以及含有要执行的应用软件的 `loop()` 循环函数）。回顾图 5-1 会发现 IDE 提供了一个有用的模板，用户只需将相应代码填入其中即可。

这些工具与软件组件为 AVR 微控制器创建可执行代码，它们可被划分为两大类：一类是主机开发工具和运行时编译源码，另一类是目标 AVR 控制器的可执行二进制代码。图 5-4 是 Arduino 软件主要组件的框图，描述用户提供的程序（sketch）如何形成一个完整的、可执行的二进制映像。

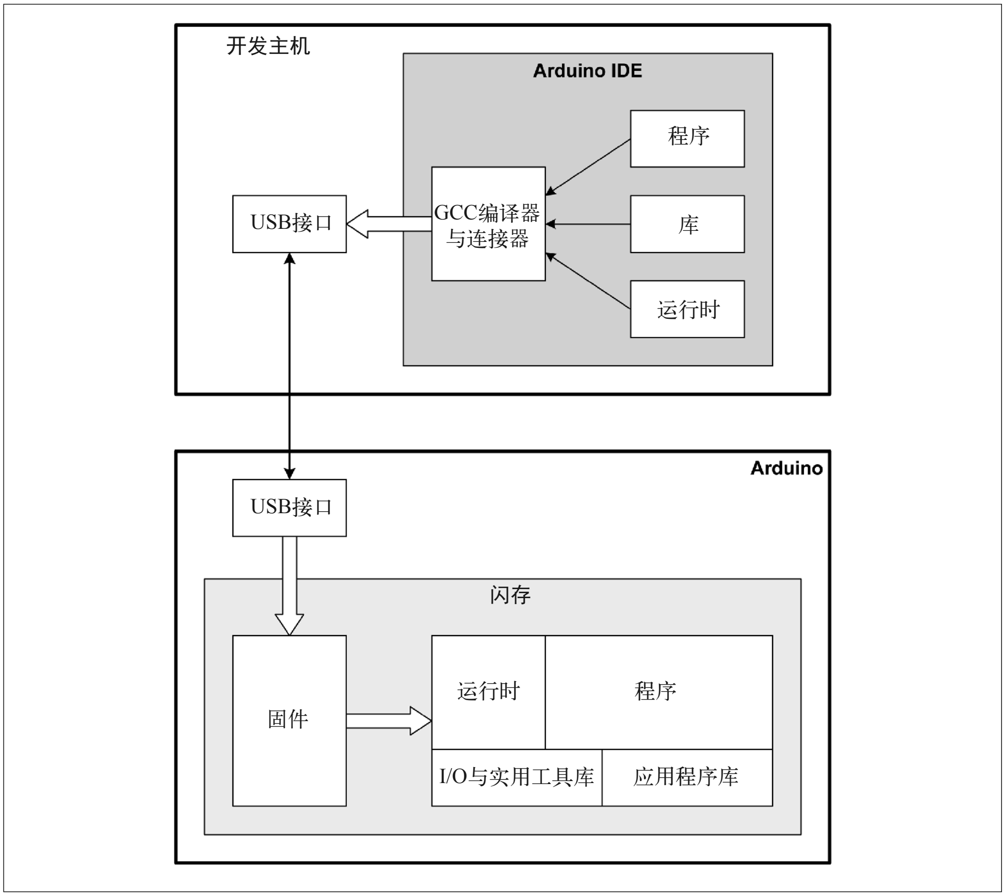


图 5-4：Arduino 软件组织

5.4.2 Arduino软件创建过程

使用 IDE 创建 Arduino 软件时，有如下 5 个主要步骤。

准备源文件

用户编写的程序被 IDE 略做修改，为 IDE 1.0 或更新版本添加 `#include "WProgram.h"`，或者为较旧版本的 IDE 添加 `"Arduino.h"`。不带扩展名的标签（源文件）与主程序串联在一起，生成一个大的源代码文件（带有 .c 或 .cpp 扩展名的标签被单独编译）。

IDE 也尝试为程序中除 `setup()` 与 `loop()` 之外的其他函数创建函数原型。它们被放置在程序文件顶部，紧接在注释或预处理声明语句（`#include` 与 `#define`）之后，但在其他语句之前。在传统的 C 或 C++ 源文件中，它们会被放到 .h 头文件之中，编译代码时将其包含进来。Arduino IDE 会帮你做这些事，它会动态生成这些函数原型，并将之插入源文件。准备源文件的最后一步是，把标准的 main.cxx 文件的内容添加到用户程序中（定义 `main()` 的地方）。

编译

Arduino 使用 AVR-GCC 编译器套件，将源代码"翻译"为名为"*目标文件*"的二进制文件。它们不能立即执行，在 Arduino 硬件处理它们之前，必须先使用"链接器"工具将其链接在一起（下一步会讲到）。AVR-GCC 编译器是在 GCC 编译器基础之上专为 Atmel 的 AVR 设备开发的。因受到目标硬件的限制（主要是内存），AVR-GCC 并不支持 C++ 语言的所有功能，但完全支持 C 语言。

编译器接收大量命令行参数，这些参数也被称为"开关"。Arduino IDE 负责为编译器提供正确的"开关"。这些"开关"设定包括标准 include 文件所在的位置、优化选项、目标专用选项、警告信息级别等。

链接

链接是将目标文件与库模块链接在一起的过程，其基本思想是填充目标文件中的"孔洞"。在那里，原始源代码引用外部库或对象中的数据对象或函数，但编译器在编译代码时将无法解析地址。链接器的工作是定位缺失引用，并把它们写入最终的二进制可执行文件，其中包括被引用数据或函数的二进制代码。

转换

链接器生成的二进制文件必须被转换为 Intel 十六进制格式文件，AVR 设备中的 BootLoader 固件会使用它。avr-objcopy 实用程序可以完成这个工作，相关内容将在第 6 章讲解。

上传

最后一步是，将制作好的二进制可执行文件传送到 Arduino 硬件。该过程通常由 USB 接口、微控制器中的 BootLoader 固件，以及名为 AVR-DUDE（这其实是一个缩略语）的实用程序完成。当然，也可以直接通过 ICSP 接口将可执行代码加载到 AVR 设备（比如没有 BootLoader 时），Arduino IDE 对此提供了支持。

5.4.3　程序标签卡

事实上，将一个大程序分割为若干个小的源代码模块（每个模块拥有自己的 include 文件），这在 IDE 中可以办到，而且相当方便。IDE 加载一个程序时，会在程序目录中查找任何附加文件。辅助文件可能没有扩展名，或者带有 .c、.cpp、.h 扩展名。虽然带有 .c 或 .cpp 扩

展名的文件出现在标签卡中，但它们会被编译到与主程序代码相链接的目标文件。不带有扩展名的文件被包含到程序。为了使用带有 .h 扩展名的标签文件，必须使用 `#include` 语句进行引用，并且要使用双引号（而非尖括号）。

如果你的辅助文件带有自己的 include 文件，那么它们必须全部包含到主程序——即使其本身并未引用任何代码。这同样适用于外部库模块。如果一个辅助文件使用了某个库中的某个类，而主程序并未使用它，那么辅助文件与主程序仍然需要引用该库的 include 文件。

我喜欢的一种做法是，将所有全局变量（包括前面提到的类对象）放入一个独立的源文件。这允许其他任何需要访问变量的模块这样做。这也使得主程序更易读，因为主程序中只包含 `setup()` 与 `loop()` 两个函数。在 Arduino.cc 站点，你可以读到更多关于多文件程序的内容（http://bit.ly/arduino-build-process）。第 11 章还会介绍关于 DDS 信号发生器的示例，你可以在 Github（https://github.com/ardnut）上找到完整可用的代码。

5.4.4 Arduino软件架构

无论大小，任何一个 Arduino 程序至少由 `setup()` 与 `loop()` 两个函数组成。程序启动时，只调用执行一次 `setup()` 函数，而 `loop()` 函数会一直执行，直到设备断电或者 Arduino 开发板重置。`loop()` 函数被 `main()` 函数反复调用，`main()` 函数由 Arduino IDE 自动提供。当然，程序中也可以包含其他更多函数。比如在第 12 章温控器程序中，除了 `setup()` 与 `loop()` 两个必需的函数外，还有其他多个函数。有关 `main()` 函数运行时代码的讲解，请参考 5.4.5 节。

程序文件的开始部分也包含一些声明语句，它们用于包含其他文件，为 I/O 引脚与其他值定义常量以及全局变量。在 5.4.6 节，你可以看到一个标准的 Arduino 程序是如何组织的。在 5.4.7 节和 5.4.8 节，我们将分别讨论常量与全局变量。

有时，永不停息的 `main()` 循环也称为“*事件循环*”（event loop），它是为微控制器进行编程的常见方法。类似 AVR 的这种小型设备不会从磁盘中加载程序文件，并且其上运行的任何程序都被设计成反复执行一个特定的功能（或某些功能）。因此，使用“永远运行的主循环”这一说法很有道理。因为程序代码存储在闪存中，当 Arduino 通电或重置后，程序代码就准备好再次运行。

为了让各位有个正确的认识，图 5-5 显示了 Arduino IDE 提供的 `main()` 函数在程序中如何调用 `setup()` 与 `loop()` 函数。

图 5-5 中，在 `setup()` 与 `loop()` 函数外面，将一个外部库的类对象进行实例化（初始化）。所以对于程序中的所有函数，它都是可用的。除了 `setup()`、`loop()`、主程序中的其他任何函数外，其他文件中也可以有函数，并且程序中包含的其他源代码文件将被 Arduino IDE 作为标签进行处理。

请注意，标签并非真正的库，它只将更多程序源码放入单独的文件。标签中的代码可以在其他程序中重用，但与真正的库不同，创建程序时，你需要手动指定标签，而不能只在主程序的源文件中使用 include 指令。为了以标签形式将代码添加到程序，在 IDE 菜单栏中，依次选择“项目 - 添加文件”。所添加的标签源文件不必非得与主程序文件在相同目录下。

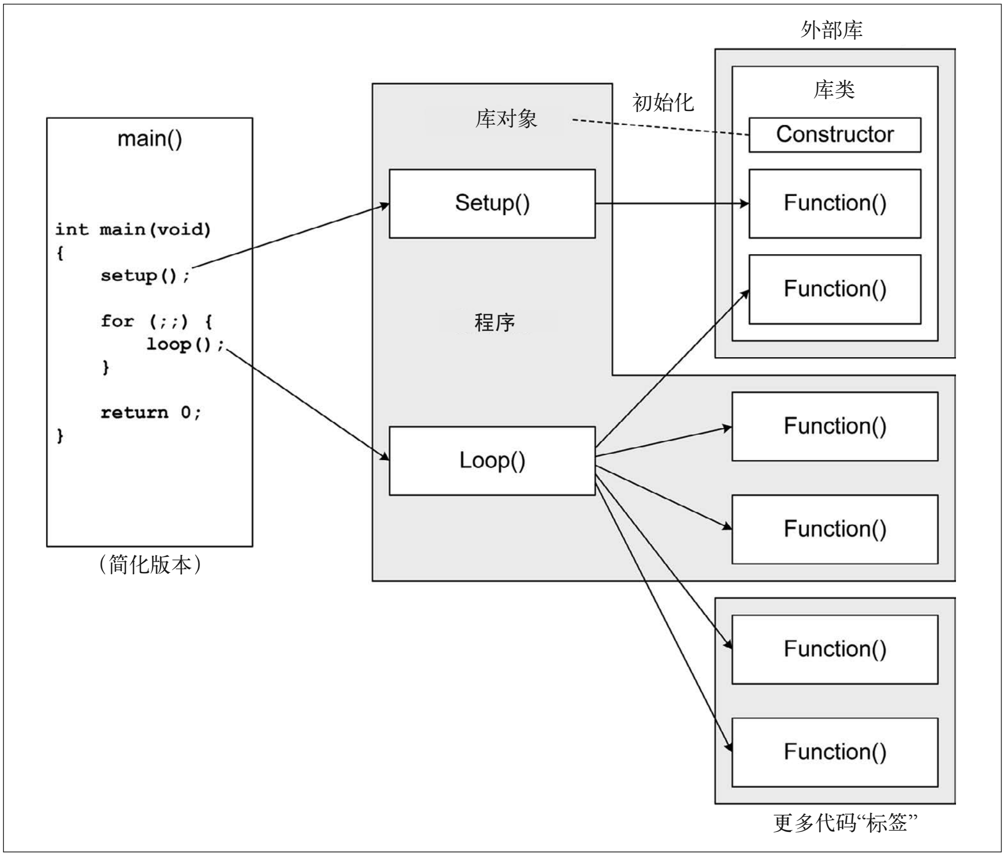


图 5-5：Arduino 程序结构

当程序代码与其他标签文件一起编译后，外部库与最终可执行映像集成在一起。Arduino IDE 将根据程序开始部分的 include 语句生成正确的链接器选项。关于如何将库集成到程序中的更多内容，请阅读 5.5.1 节。

5.4.5 运行时支持：`main()`函数

Linux、Windows、Mac OS X 系统使用的编译器提供了针对特定平台的函数库，称为“运行时库”。在基于 GCC 的系统中，一般称为 libgcc.a 或 libgcc_s.so.1；而在 Windows 系统下，运行时库通常称为 MSVCRT.DLL（C 语言）与 MSVCPP.DLL（C++）。“运行时库”的 AVR 版本为 avr-libc，相关内容将在第 6 章讨论。运行时库包含着特定于某个平台的常用函数，比如常见的数学运算、底层 I/O 函数、系统计时器、`printf()` 支持等。

Arduino 也有自己的运行时支持模块，不过稍有不同。程序本身不会做太多事情，它没有用于启动的 `main()` 函数，启动后也无法连续执行。Arduino 运行时支持包含必要的 `main()` 函数，以及其他启动配置函数。如示例 5-4 所示，`main()` 函数其实非常简单，它是小型微控制器的典型设计。

示例 5-4　Arduino `main()` 函数

```
int main(void)
{
    init();
    initVariant();

    #ifdefined(USBCON)
    USBDevice.attach();
    #endif

    setup();
    for (;;) {
        loop();
        if (serialEventRun) serialEventRun();
    }
    return 0;

}
```

根据 IDE 中所选的 Arduino 目标硬件类型，在编译期确定对 `init()`、`initVariant()` 与 `USBDevice.attach()`（如适用）的调用。针对每种 Arduino 开发板，这些构成了运行时支持代码的主要组成部分。在更大的实时操作系统（RTOS）中，它们可能是 RTOS 厂商或开发者提供的开发板支持包（BSP）的一部分。下面分别介绍这几个函数。

`init()` 函数

　该函数位于 Arduino/hardware/arduino/avr/cores/arduino/wiring.c，根据 Arduino 开发板上使用的 AVR 部件初始化各种 AVR 外围元件，比如定时器、定时器 / 计数器预引比例因数、A/D 转换器预引比例因数、PWM 输出模式。

`initVariant()`

　该函数以弱函数声明的形式提供“钩子”，通常用于为硬件提供额外的运行时初始化。对于这部分初始化，标准的 Arduino 开发环境并未提供相关定义与代码。

`USBDevice.attach()`

　`attach()` 是 `USBDevice` 类的一个方法，在 Arduino/hardware/arduino/avr/cores/arduino/USBCore.cpp 文件中，你可以找到 `USBDevice` 类，它提供了与 USB 接口进行通信的必需功能。

对许多人来说，这些初始化函数所做的工作与他们无关。然而，如果你真的想了解 Arduino IDE 的工作及其工作原理，那么阅读这些支持函数的源代码是非常值得的。5.6 节将介绍如何获取源代码。

如前所述，程序中的 `setup()` 与 `loop()` 函数要由开发者编写。`setup()` 函数通常用于定义指定的 I/O 端口与初始状态，对一些 AVR 外围功能进行初始化，或者执行其他一次性操作。`setup()` 函数多半是可选的，甚至可以是空函数，但它必须在程序中存在，否则链接器就会报错。

`loop()` 函数包含着程序的主要活动。正如在 Arduino `main()` 函数中看到的，`loop()` 函数会被反复调用执行，直到 Arduino 开发板断电。`loop()` 函数执行期间允许发生中断，一些应

用可以集成定时器中断或延时，让 loop() 函数的执行呈现某种周期性（与简单的自由运行相反）。

5.4.6 程序示例

正如我们所看到的，一个基本的 Arduino 程序由两部分组成：setup() 函数与 loop() 函数。由于 main() 函数由 IDE 提供，所以编写简单程序时，我们不必担心它。

示例 5-5 是一个简单的 Arduino 程序，用于监控几个输入。当其中一个输入从低电平变为高电平时，产生一个输出。你可以以它为基础，制作一个简单的防盗警报器。

示例 5-5 简单的防盗警报器

```
// 常量
// 定义要用的数字I/O引脚
#define LOOP1   2       // 感知循环1
#define LOOP2   3       // 感知循环2
#define ARM     6       // 报警器
#define RESET   7       // 重置报警器状态
#define ALERT   12      // 报警器输出
#define LED     13      // 板载LED灯

#define SLEEPTM 250     // 循环暂停时间

#define SLOWFLASH   10  // 解除闪光
#define FASTFLASH   2   // 准备闪光

// GLOBAL VARIABLES
// State flags
bool arm_state;         // T=待命,F=解除待命
bool alarmed;           // T=警报状态 F=静默
bool led_state;         // 控制板载LED灯
bool loop1state;        // 感知循环1的状态
bool loop2state;        // 感知循环2的状态
bool in_alarm;          // T=启动警报 F=静默

int led_cnt;            // 闪光循环计数器
int flash_rate;         // 计数因子

// 我们将使用Arduino内置的pinMode()函数设置
// 数字I/O引脚的输入或输出行为

void setup()
{
    // 初始化引脚2、3、6、7为数字输入
    pinMode(LOOP1, INPUT);
    pinMode(LOOP2, INPUT);
    pinMode(ARM, INPUT);
    pinMode(RESET, INPUT);

    // 初始化引脚12、13为数字输出
    pinMode(ALERT, OUTPUT);
    pinMode(LED, OUTPUT);
```

```
    // 初始化状态标识
    arm_state  = false;
    alarmed    = false;
    led_state  = false;
    loop1state = true;
    loop2state = true;
    in_alarm   = false;

    // 设置LED起始条件
    led_cnt = 0;
    flash_rate = SLOWFLASH;
    digitalWrite(LED, LOW);
}

void loop()
{
    // 获取警报开关值
    arm_state = digitalRead(ARM);

    // 若reset为低电平(真),则取消警报状态
    if (!digitalRead(RESET)) {
        in_alarm = false;
}

    // 若未待命,则暂停循环
    if (!arm_state) {
        flash_rate = SLOWFLASH;
        in_alarm = false;
    }

    else {
        flash_rate = FASTFLASH;
    }

    // 检查感知循环s
    loop1state = digitalRead(LOOP1);
    loop2state = digitalRead(LOOP2);

    // 若已待命,则进入警报状态
    if (arm_state) {
        if ((loop1state) || (loop2state)) {
            in_alarm = true;
        }
    }

    if (in_alarm) {
        digitalWrite(ALERT, HIGH);
    }

    else {
        digitalWrite(ALERT, LOW);
    }

    led_cnt++;
```

```
        if (!(led_cnt % flash_rate)) {
            led_cnt = 0; // 重置闪光技术
            led_state = !led_state; // 反转LED状态
            digitalWrite(LED, led_state);
        }

        delay(SLEEPTM);
    }
```

图 5-6 显示如何将 Arduino 连接到门窗的外部开关。采用串联方式连接各个开关，并把链的一端连接到地。其中原因很简单：如果任何一个连接传感器开关的线路断开，就会触发报警器。

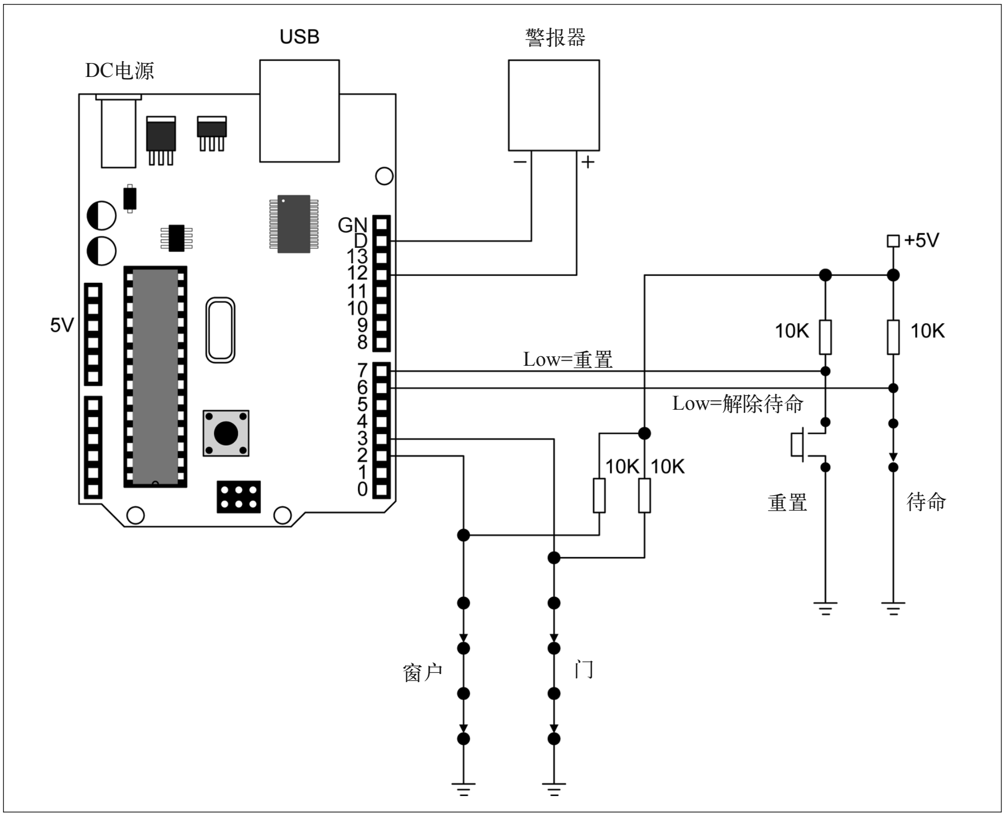


图 5-6：简单的防盗报警器

在类似的应用中，传感器开关可以是窗框栓塞、磁驱动舌簧继电器、滑动门的叶片开关，甚至可以是一个传感器模块（比如声音探测器）。警报器可以是任何一种报警装置，比如 50 美分的压电式蜂鸣器，以及控制警报器或手机自动拨号器的继电器。关于更多与 Arduino 相兼容的传感器，请阅读第 9 章。这个基本设计可以向许多有趣方向做进一步扩展，并能在其基础之上创建商业级别的警报系统。从 GitHub（https://www.github.com/ardnut）可以下载到可用的代码。

5.4.7 常量

有两种方法可以将 I/O 引脚号、时间延迟以及其他值定义为常量。第一种方法是使用 `#define` 语句，第二种方法是定义整型变量，将其初始化为某个值，并不再修改其值。比如，使用 `#define` 语句关联引脚 2 与某个名称，如下所示：

```
#define LOOP1 2  // 感知循环1
```

另一种替换方法是创建一个整型变量以保存引脚编号，如下所示：

```
int LOOP1 = 2;   // 感知循环1
```

仔细查看 Arduino 示例代码就会看到，上面两种方法都用到了，并且都能工作得很好。尽管出于某些原因，有些开发者会尽量避免使用 `#define` 语句，但从内存使用角度看，这种语句还是值得考虑的。

定义常量时，示例 5-5 使用了 `#define` 语句。这样做的好处是，它们会让编译出的程序更小。就示例程序而言，使用 `#define` 语句时，最终编译的程序大小为 1452 B。若使用 int 声明变量以取代其中的 `#define` 语句，则最终编译的程序大小为 1540 B。两者相差 88 B，看上去并没有节约多少内存，但对于拥有大量 I/O 定义与常量的大型程序，这种微小的差距将不断累加。比如，对于 ATmega328，程序必须限制在 30 720 B 以内，此时使用 `#define` 语句还是整型变量将决定程序能否加载到设备并执行。

5.4.8 全局变量

你可能注意到，所有 Arduino 程序都在使用全局变量。这在 Arduino 程序中是常见做法，原因在于，运行程序代码时，`loop()` 函数会被 `main()` 函数反复调用。如果将变量定义在 `loop()` 函数中，那么每次调用 `loop()` 函数时，这些变量都会被擦除并重新加载。如果 `loop()` 函数需要一个持久的变量以保持数值（比如计数器），那么需要将该变量定义在 `loop()` 函数之外，以防止它被反复清除与重建。全局变量也为 `setup()` 函数在 `loop()` 函数被首次调用之前设置初始值提供便利，并且程序中的其他函数也能访问这些变量。

在某些场合，全局变量的“名声”可不怎么样，因为程序的不同部分之间可能产生意外的“耦合”（coupling）问题。在从磁盘载入的大型程序中，先执行，然后被用户终止，比如文字处理程序、游戏、网页浏览器，使用全局变量是合情合理的。但当程序的一部分修改一个全局变量时，程序的另一部分也在修改同一个全局变量，这会导致整个程序崩溃。在某些系统中，它拥有大量可用内存、动态内存分配、信号量和互斥锁，能够把指针传递到复杂结构，此时使用全局变量也许并不合理。

然而，在嵌入式系统领域，全局变量通常被用作一种共享内存的高效形式，此时全局变量对程序中的每个函数都是可见的。你可以把全局变量看作布满各种开关、指示灯、旋钮、表盘的面板，就像飞机的驾驶舱。只要遵循“一个函数修改，多个函数读取”的规则，一切都不会有问题。

由于 Arduino 程序不会以线程或进程的形式并行运行，所以不存在访问冲突问题。中断可能是个棘手的问题，这取决于如何实现，而一点点深谋远虑也可以消除潜在的危险。

5.5 库

Arduino IDE 提供的库涵盖范围很广，但也不是无所不包，相关内容将在第 7 章讲解。除非有人投入时间与精力为特定传感器或接口创建库，否则需要你自己编写相应代码。当你使用定制或非常见的传感器或扩展板（需要使用专用函数）时，经常需要自己亲自动手编写相应代码。

Arduino IDE 中使用的术语“库”，似乎与在大型项目中使用 GCC 或 Visual Studio 时所用的“库”有点出入。为全尺寸计算机（比如桌面 PC）创建软件时，人们所说的“库”通常指二进制对象文档。在 Linux 与 Unix 系统中，这可以通过使用 ln、ar、ranlib 工具实现。生成的二进制文件是一系列带有索引的目标文件（见 5.1 节中的“目标代码、映像和源代码”部分），链接器用于填充“间隙”并生成完整程序。如果是动态链接库，应用程序启动时（甚至在程序运行期间）就会加载库，同时进行链接。

在 Arduino IDE 中，库通常作为源代码存在，直到程序需要它们。所以，编译程序时真正发生的是，程序（以及相关标签文件）、所需要的库、运行时代码都被同时编译，并被链接到一个单独的二进制可执行映像。

5.5.1 在Arduino程序中使用库

如前所述，当 Arduino IDE 遇到一个 include 语句，并引用一个已经向 IDE 注册的库时，IDE 将生成必需的创建步骤与链接器选项，以便将库自动包括在内。为此，IDE 必须事先知道相关库。

关于向 IDE 注册库的内容，5.5.2 节将进行讲解。对于常见的操作和预装的 I/O 设备，Arduino IDE 提供了一系列库。图 5-7 列出一个较旧版本的 IDE 中包含的可用库。图 5-8 列出了 IDE 1.6.4 版本包含的函数库与库管理选项。

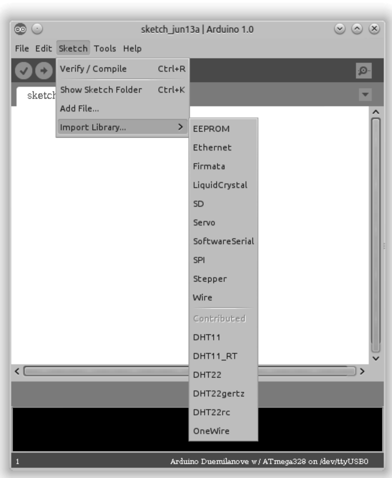


图 5-7：较旧版本 IDE 中注册的库

示例 5-6 演示如何在 Arduino 程序中包含并使用函数库（此处是 SoftwareSerial 库）。首先要注意位于程序文件顶部的包含语句。

```
#include <SoftwareSerial.h>
```

由此，IDE 知道它需要定位一个名为 SoftwareSerial 的库，并且希望这个库位于自己已知的库模块列表中。

示例 5-6　库示例 1

```
#include <SoftwareSerial.h>

SoftwareSerial softSerial(2, 3);

int analogPin = A0; // 为电位器选择输入引脚
int analogVal = 0;  // 该变量用于保存来自传感器的值

void setup()
{
    // 为软串口设置数据传输速率
    softSerial.begin(9600);
}

void loop() // 反复运行
{
    // 从模拟输入读一些数据
    analogVal = analogRead(analogPin);

    // 写数据到软串口
    if (softSerial.available())
    softSerial.write(analogVal);

    // 等待1 s（1000 ms）
    delay(1000);
}
```

使用如下代码行对 SoftwareSerial 对象进行实例化。

```
SoftwareSerial softSerial(2, 3);
```

以上代码同时也将数字 I/O 引脚 2 与 3 用作串行通信。

在 `setup()` 函数中，设置串口通信的波特率。

`loop()` 函数先从 A0 逻辑输入引脚读取值，然后通过 `softSerial` 对象发送二进制值。`loop()` 函数在每次发送完二进制值之后暂停 1 s。

这一技术能在两个 Arduino 开发板之间收发二进制数据，但如果你想要人类易读的输出，那么它工作得不是很好。为此，你需要使用函数库中的 `print()` 与 `println()` 方法，如示例 5-7 所示。

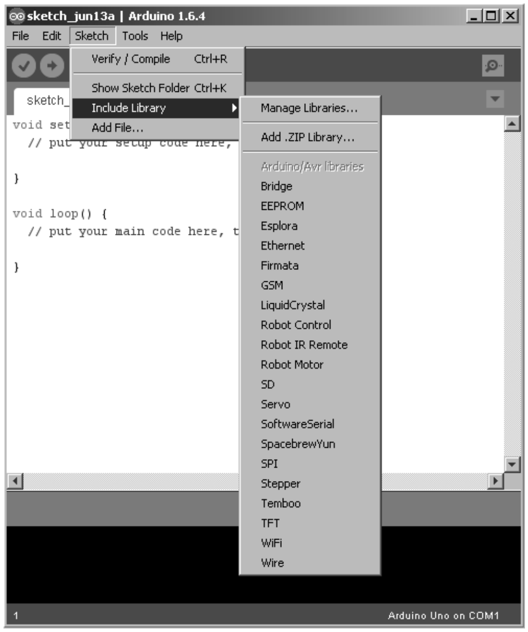


图 5-8：Arduino IDE 1.6.4 中的库列表

示例 5-7　库示例 2

```
#include <SoftwareSerial.h>

SoftwareSerial softSerial(2, 3);

int analogPin = A0; // 为电位器选择输入引脚
int analogVal = 0;  // 用于存储来自传感器的值

void setup()
{
    // 为软串口设置数据速率
    softSerial.begin(9600);
}

void loop()
{
    // 从模拟输入读取一些数据
    analogVal = analogRead(analogPin);

    // 写数据到软串口
    softSerial.print(analogVal);
    softSerial.println(); // 打印一个换行符

    // 等待1 s（1000 ms）
    delay(1000);
}
```

也可以使用标准库中的 `printf()` 函数将数据发送到串口，但 `printf()` 函数默认不随 Arduino 核心函数一起启用。如果想了解如何启用 `printf()` 函数，并且理解它如何影响自己的程序，请阅读 Arduino 官方站点中有关启用 `printf()` 函数的页面（http://playground.arduino.cc/Main/Printf）。

在 IDE 集成的帮助中，可以找到有关 Libraries（库）的内容，从中可以学到更多关于 SoftwareSerial 库模块的知识。并且，也可以从 Arduino 安装目录中找到源代码。有关 SoftwareSerial 的内容将在第 7 章详细讲解。

5.5.2 将库添加到Arduino IDE

用户提供的扩展库是使用外部设备的关键，这些外部设备包括湿度和温度传感器、射频模块（RF）、红外遥控组件（IR）。你可以自己编写函数库，也可以使用其他人编写好的函数库。

扩展库通常安装在 sketchbook 目录下名为 libraries 的子目录中。包含扩展库的文件集位于子目录，并且带有合适的名称。该子目录名称出现在 IDE 的函数库列表之中。有两种方法可以将一个库添加到 Arduino IDE。

方法 1：自动

在 Arduino IDE 1.0.5 及以后版本中，IDE 可以自动为你将函数库源代码安装到 sketchbook 目录之下（具体位置取决于所用的操作系统）。在 IDE 主菜单中，依次点击 Sketch → Import Library → Add Library，然后选择包含库源码的目录或 ZIP 文件。在 Arduino IDE 1.6.4 中，添加库的步骤略微有些不同，库管理功能被划到 Include Library 菜单之下。

这种方法的一个好处（除了简单之外）是，IDE 可以立即识别新库，并且不需要重启。此外，使用新版本的 IDE 可以从外部源下载并安装库。

方法 2：手动

为 Arduino IDE 手动安装扩展库遵循如下步骤。

- 首先关闭 IDE（如果它处于运行状态）。
- 将新的库文件解压到临时目录（大部分扩展库都以 ZIP 文件形式存在）。
- 应该看到至少两个源文件：.c 或 .cpp 文件（程序）、.h 文件（包含文件或头文件）。
- 在 <home>/sketchbook/libraries 目录下创建一个子目录，名称与两个源文件相同。
- 将临时目录中的所有内容复制到上一步创建的新目录中。

再次启动 IDE 时，在“项目 -Include Library”下的子菜单中，应该能够看到手动添加的库。

扩展库的内容组织约定如下：

```
IRTracker/IRTracker.cpp
         /IRTracker.h
         /keywords.txt
         /README
         /utility/IRCal.cpp
                 /IRCal.h
```

```
                  /IREncode.cpp
                  /IREncode.h
                  /IRDecode.cpp
                  /IRDecode.h
/Examples/ScanIR/ScanIR.ino
                 /SendStop/SendStop.ino
```

请注意，上面只是一个示例。据我所知，对于 Arduino，目前仍未有可用的 IR tracker 库。

库的子目录名称必须与源文件一样，这种约定适用于整个 Arduino IDE 目录。比如，在 Arduino 示例目录（Linux 系统中为 /usr/share/arduino/examples）中，子目录拥有与 .ino 文件一样的名称。

正如你在示例目录结构中所看到的，在库的主目录之下，除了与库目录拥有相同名称的两个文件之外，还有包含其他实用函数的子目录以及源代码文件。虽然它们不会出现在库列表中，但编译时，库代码会访问它们。

你可能已经注意到，库的根目录之下有一个名为 keywords.txt 的文件。它非常重要，为 IDE 提供一些关于库源代码的简单描述。示例中，IR tracer 库的 keywords.txt 的内容大致如下所示：

```
#  IR  Tracker
##################################
MAXPWR      LITERAL1
MAXTIME     LITERAL1

LastCal     KEYWORD1
TotHours    KEYWORD1

IRState     KEYWORD2
IRSense     KEYWORD2
```

文件结构非常简单，# 表示行注释，紧接其后的单行内容会被忽略。表 5-1 列出的特殊关键字用于向 IDE 指明字面常量、类、结构体、变量、函数类型。

表5-1：keywords.txt中的关键字定义

关键字	定义
LITERAL1	字面常量（如宏）
LITERAL2	字面常量（如宏）
KEYWORD1	类、数据类型、C++ 关键字
KEYWORD2	方法与函数
KEYWORD3	结构体

在你的系统中安装 Arduino 运行时组件的目录下，keywords.txt 文件中，也可以发现更多信息。在我的 Linux 机器中，该 keywords.txt 文件位于 /usr/share/arduino/lib 目录；而在 Windows 系统中，可以在 C:\Program Files\Arduino\lib 目录中找到它。也可以查看 IDE 提供的库子目录中的 keywords.txt 文件，以了解别人做了什么。

当然，你也可以把扩展库放入 Arduino IDE 预定义的目录，但我不建议这样做。因为将 IDE 升级到一个更新的版本时，其提供的预定义库可能发生变化。用户提供的库与程序从来不应该被升级所改变。

不管怎样，一个库组件由两个基本部分组成：源模块（name.c 或 name.cpp）与包含文件（name.h）。源模块中的代码是一系列函数，它们组成一个简单的 C++ 类。类本身定义在头文件中。要想使用库，只需将其复制到相应目录下并启动 Arduino IDE，然后从菜单栏中依次选择“项目 -Import Library”即可。IDE 会检查头文件，并将 `#include<name.h>` 语句插入用户程序。

5.5.3 创建自定义库

在 Arduino IDE 中创建自定义库很简单，但这并不意味着我们只能创建简单的库，因为有些库可以很复杂。但无论简单还是复杂，它们都是按照基本模板创建的。源代码使用 C++（AVR 受限版本）语言被编写为一个类，可能还带有一个或多个相关类，以提供辅助。第 7 章列出了一些库的源代码，浏览它们可以了解库的源代码是如何组织的。

如前所述，一个最小的库至少由 .cpp 源文件与 .h 包含文件组成。其中，源文件包含库类的实现，包含文件含有类定义、类型定义以及宏定义。

回忆一下前面库示例的目录结构，包含库源代码与包含文件的目录与主源文件拥有相同名称。库目录下也应该包含 keywords.txt 与 README 文件，而包含 examples 目录是一个好的主意（尤其是你打算发布这个库以供他人使用时）。

5.6 Arduino源代码

对于基于 AVR 的开发板与基于 ARM 的 Due 开发板（本书不做讲解），完整的 Arduino 源代码集合中包含着它们的源文件。可以通过 URL 与 `git clone` 命令，从 GitHub（https://github.com/arduino/Arduino.git）获取这些源代码。当然，也可以直接从 GitHub 下载包含源代码的 ZIP 文件（https://github.com/arduino/Arduino/archive/master.zip）。

如果打算深入研究 Arduino 而不局限于 IDE，那么在手头准备一份源代码是非常有用的。阅读源文件能够帮助你更清楚地了解底层细节。使用 Doxygen（http://bit.ly/doxygen-main）等工具时，能够根据依赖关系图、调用关系图、类索引、源文件与函数创建一系列带有链接的 Web 页面。虽然 Arduino 源代码没有太多 Doxygen 专用标签，但你仍然可以使用它创建有用的文档。

从高层看，Arduino 源文件目录由 app、build、hardware、libraries 等几个子目录组成。其中，app 目录存放应用程序源代码，build 目录存放编译模块，hardware 目录存放硬件专用文件，libraries 目录存放各种库模块，它们包含在标准的 Arudino 发行版中。从底层视角看，hardware 与 libraries 两个目录是最“有趣”的。

在 hardware 子文件夹中能找到各种源代码，比如各种 BootLoader、运行时支持代码（在源文件集合中称为 core，包含 main.cpp），以及一些与 EEPROM、serial I/O、SPI、双线接口

（位于 hardware/avr/arduino/libraries 目录，不同于前面提到的 libraries 目录）相关的模块。Arduino IDE 支持的库位于 libraries 目录，其下子目录包含声音、以太网、液晶显示、SD 存储卡、servo 发动机、步进电机、TFT 显示、WiFi 模块相关的源代码，包含针对 Arduino Robot、Esplora、Yún 产品的函数库。许多库文件夹同时包含有示例程序，有些还有类似于文本文件的文档。

阅读 Arduino 源代码时需要注意，其中使用了大量的 `#if`、`#ifdef`、`#ifndef`、`#define` 语句，以决定为指定的 Arduino 开发板包含以及编译哪些代码。初次接触时，这多少会让人感到困惑，可能需要花费一些努力才能搞清楚究竟是怎么回事。另外值得注意的是，某些情况下会使用一个或一组函数以实现特定用途，而另外一些情况下会定义一个类以达到相同目的。

第6章

不使用Arduino IDE编程

其实，Arduino 硬件没什么特别的，它只是一个基于 Atmel AVR 设备的非常基本的开发板。而 Arduino IDE 与 BootLoader 固件让非编程人员也能够很容易地使用，并让一切工作正常。不过，你也可以完全不用 Arduino IDE 为 Arduino 开发板编程。Arduino IDE 的确是个使用方便的应用程序，它能帮助开发者“打理”软件编译过程中的大量混乱细节，但喜欢使用命令行工作的开发者只使用一个简单的文本编译器也能做到，完全不必使用 Arduino IDE。

本章将通过一些示例学习为 Arduino 创建程序的其他方法，还要学习如何在命令行中使用 AVR-GCC 工具链，并且只用 makefile 而不借助其他工具。此外，也要了解如何使用汇编语言对底层编程（down to mental），以从 AVR MCU“榨取”最后一点性能。

为 Arduino 创建可执行代码的方法不止一种，同样，将软件上载到 AVR 的方法也有很多。本章将学习如何使用其他方法完成这项工作，而不使用 Arduino IDE。

6.1 IDE替换方案

为 Arduino 开发板上的 AVR MCU 开发并加载程序时，Arduino IDE 并不是唯一的方法。为 Arduino 编程的一个替代方法是使用 PlatformIO 工具，该工具可以运行在 Linux、Mac OS X、Windows 平台下。PlatformIO 是一个基于 Python 的代码构建器与库管理器，它从命令行执行。另一个用于创建 Arduino 程序的 Python 工具是 Ino 工具，它工作在 Linux 与 Mac OS 系统下，目前尚不能在 Windows 环境下运行。

6.1.1 PlatformIO

PlatformIO（http://platformio.org）是一个基于 Python 的命令行工具，它支持超过 100 种

目标微控制器。并且提供跨平台支持，只要有 Python 2.6 或 2.7 环境，它就能正常运行在 Windows、Mac OS X 或 Linux 平台下（我推荐各位安装 Python 2.7）。更多相关信息请访问 PlatformIO 官方站点。

根据指定微控制器的类型，PlatformIO 会确定需要哪些工具链组件，然后以正确的顺序调用它们并完成编译、链接与目标上传操作。PlatformIO 支持的开发板包括 Adafruit 推出的 Trinket 系列、所有 Arduino AVR 开发板、BitWizard Raspduino（第 1 章）、Digistump 的 Digispark 开发板、Engduino、LowPowerLab Mote 开发板、Microduino、Sanguino AVR 开发板，以及来自 SparkFun 的 AVR 开发板。它也支持各种基于 ARM 的产品，比如使用 Atmel SAM MCU、STM32 MCU、LPC（NXP）、Nordic nRF51 以及 TI MSP430 系列等的开发板。

在 Linux 或 Mac 系统中安装 PlatformIO 非常简单。你可以使用命令行下载安装脚本，或者使用 Python pip 工具。可能需要先安装 cURL 数据传送工具，但这是一种常见的实用程序。在 Linux 系统中，可以直接从软件包存储库获取。

成功下载并安装 PlatformIO 包之后，控制台的输出如图 6-1 所示。

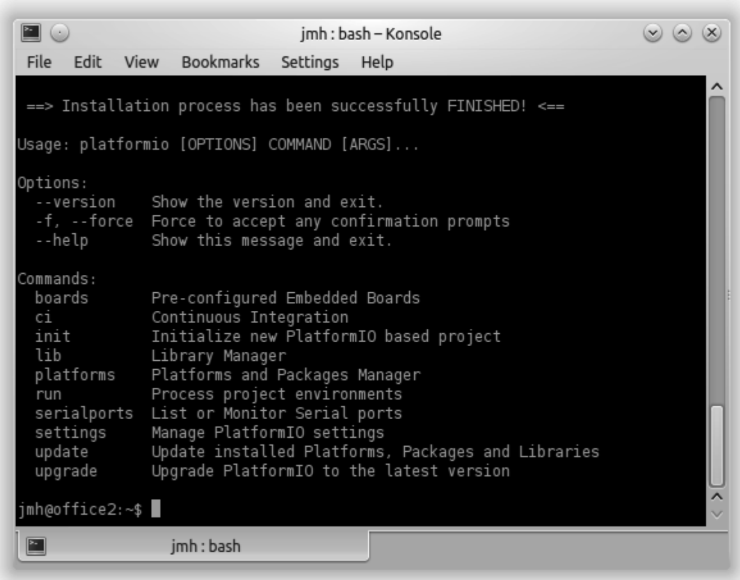


图 6-1：成功安装 PlatformIO 后的控制台输出

一旦安装好 PlatformIO 包，你就能搞清它都能做些什么。输入 `platformio boards` 将看到一个长长的列表，显示当前支持的开发板。

PlatformIO 使用“项目”这一概念来保持代码文件简洁整齐。每个项目都包含一个配置文件与两个子目录（src 与 lib）。配置文件在使用之前必须先进行编辑，它是一个 ASCII KVP（键值对）INI 类型的文件。在官方站点的“项目配置文件”页面（http://bit.ly/platformio-pcf）能找到与配置文件相关的更多细节。

我建议你为 PlatformIO 项目创建一个单独的文件夹，使之与 Arduino IDE 使用的 sketchbook 目录分开。我个人将文件夹命名为 platformio（没什么创意，但适合我）。为了创建新项目，输入 `platformio init`。创建好项目之后，控制台的输出如图 6-2 所示。

PlatformIO 拥有很多功能，比如针对各种开发板的预定义框架、库管理器，它还有能力集成 IDE 或类似 IDE 的编辑器，比如 Arduino（还有 Eclipse、Energia、Qt Creator、Sublime Text、Vim、Visual Studio）。

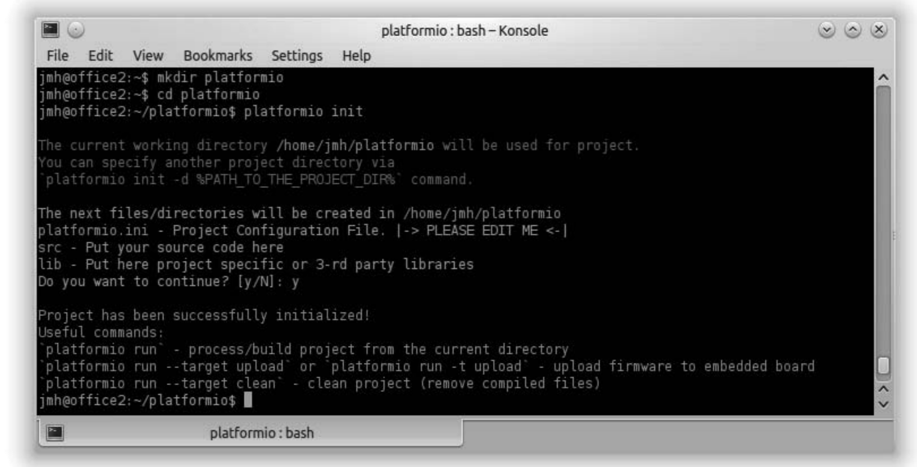


图 6-2：初始化一个新的 PlatformIO 项目

6.1.2 Ino

Ino 是一个简单的命令行构建工具，它基于 makefile 为 Arduino 目标设备进行编译。与 PlatformIO 类似，Ino 使用 Python 语言编写，但与 PlatformIO 不同，它只适用于 Arduino 开发板。Ino 支持 *.ino 与 *.pde 程序文件，也支持 *.c 与 *.cpp 源文件，并且声称支持所有 Arduino IDE 支持的开发板。请注意，目前 Ino 只工作在 Linux 与 Mac OS X 主机平台下，并且需要安装 Python 2.6 或 2.7。

可以通过两种方式下载并安装 Ino，一是下载压缩的 TAR 文件（http://inotool.org/#installation），从 GitHub 克隆即可；二是使用 Python pip 或 easy_install 工具。我使用 pip 安装 Ino 及其各种组件。安装完成后，运行 `ino -h` 命令，控制台窗口中的输出结果如图 6-3 所示。

Ino 创建并使用 makefile 文件，但它们对用户是透明的。与 PlatformIO 类似，Ino 使用基于目录的项目方案。初始化项目时，Ino 会创建 src 与 lib 两个子文件夹。它也会在 src 目录中创建一个最小的模板程序（sketch.ino），就像最新版本的 Arduino IDE 所做的那样。然后由你填充空白部分，并提供其他必需的文件。

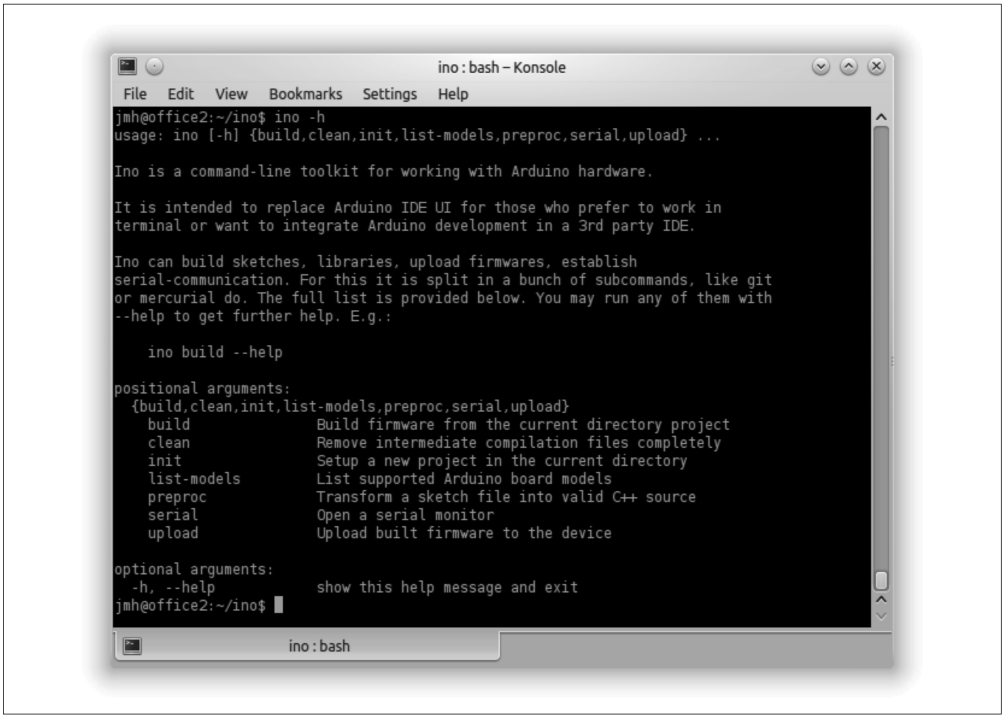


图 6-3：Ino 工具的帮助命令

可以在官方站点（http://inotool.org/）查看关于 Ino 的更多信息，也可以从 Python Package Index（http://bit.ly/ino-ppi）下载 Ino 工具。

6.2 AVR工具链

AVR-GCC 编译器及其实用工具套件是将 C 或 C++ 源代码文件转换为二进制目标文件（随后目标文件被合并到最终的可执行 AVR 程序）的主要手段，这些工具统称为“工具链”（toolchain）。如前所述，Arduino IDE 的主要作用是为这些工具提供一个用户友好的“包装壳”，并尽可能隐藏繁琐的细节。PlatformIO 与 Ino 也在 Python 脚本背后隐藏了工具链，但无论如何，它们始终在那里。如果你想使用 makefile 从命令行构建 AVR 代码，或者想手动执行每一个步骤，编译器、链接器、汇编器以及其他工具都是可用的。

> Arduino IDE 安装包（任何适用于你的 OS 的安装包）会帮助你安装 AVR-GCC 工具链。在 Linux 系统下，Arduino IDE 需要 AVR-GCC 工具链及其相关组件。因此安装 Arduino IDE 时，包管理器会同时安装它。安装工具链的主要原因是，除了 Arduino IDE 提供的工具之外，还可以使用最新版的工具。在 Windows 系统中，Arduino IDE、工具、库将被放入单独的目录，与 WinAVR 套件放置其工具链组件的位置进行区别。这样就有两套工具可以用，你可以根据需要选择。

安装 Arduino 发行软件包之后，在 Linux 系统中，你可以找到表 6-1 所示的 AVR 工具。安装好 Arduino IDE 后，在 Windows 或 Apple 系统中，你也会看到相同的工具集。关于在各种主机系统中获取与安装 GNU AVR 工具的方法，将在 6.2.1 节进行介绍。

为 Arduino 开发板上的 AVR 芯片构建可执行程序时，表 6-1 列出的工具并非都会用到。除了 AVRDUDE 之外，它们都是 GNU 工具的 AVR 版本，拥有类似或完全相同的功能。

工具链中，负责为 Arduino 或任何 AVR MCU 进行编译处理的重要工具有 avr-gcc、avr-g++、avr-ar、avr-as、avr-ld、avr-objcopy、avr-ranlib、AVRDUDE。

表6-1：AVR交叉编译工具

工具	描述
avr-addr2line	将地址转换为文件名、行号
avr-ar	生成目标代码文档、修改或提取代码
avr-as	AVR 的便携式 GNU 汇编器
avr-c++filt	对 C++ 符号进行解码
avr-gcc	用于生成 AVR 目标代码的 GCC 编译器
avr-g++	用于生成 AVR 目标代码的 G++ 编译器
avr-ld	AVR 目标代码的 GNU 链接器
avr-nm	列出嵌入在目标文件中的符号
avr-objcopy	复制与翻译目标文件
avr-objdump	显示目标文件信息
avr-ranlib	为库文档生成索引
avr-readelf	显示 ELF 文件信息
avr-size	列出目标文件 section 的大小与总大小
avr-strings	打印二进制文件中的可打印字符串
avr-strip	从 AVR 目标文件中丢弃符号
AVRDUDE	为各种 AVR MCU 编程者提供驱动程序

工具链的核心是编译器，包含 avr-gcc 或 avr-g++（针对 C++ 源码）。这些版本的 GNU 编译器是专门为 C 或 C++ 源代码，以及 Atmel 系列 AVR 微控制器定制的。它们与完整版本的 gcc 类似，只是带有一些编译选项，以方便使用它们为 AVR MCU 进行交叉编译。

浏览工具链组件的安装目录，将看到如下内容：

```
avr-c++
avr-cpp
avr-g++
avr-gcc
avr-gcc-4.5.3
```

它们只是 GNU 编译器的变体。其中，avr-c++ 与 avr-g++ 完全相同，avr-cpp、avr-gcc、avr-gcc-4.5.3 也相同。如果你感觉好奇，可以使用 `avr-gcc -v` 与 `avr-g++ -v` 命令获取版本相关信息。

源代码经过编译之后，链接器（avr-ld）会将所有二进制模块合并成一个单独的可执行文

件。使用链接器处理源代码模块之前，必须先将其编译为目标文件。

不过，编译与链接并不是整个流程的最后一步。在链接器生成二进制可执行映像文件后，这些文件还必须被转换成 Intel Hex 文件，其中包含二进制代码的 ASCII 表示形式，即 ASCII 十六进制字符。然后再上传到目标开发板，由 BootLoader 负责将 ASCII 十六进制码转换为二进制值，并将其写入微控制器的闪存。

GNU 工具链的其他成员（比如 ar、ranlib、nm、ld、strip）也有相应的 AVR 版本。图 6-4 描述了编译器、链接器、转换器、上传器如何编译程序，从库模块中链接必需的函数，然后将最终程序传送到 AVR 目标设备。

图 6-4 显示的目标文件可能是从项目的其他模块编译得到的代码，也可能是由传感器或其他配件提供的代码，还有可能是 AVR-GCC 运行时支持代码，如 Arduino IDE 提供的 `main()` 函数。其中的库对象可以是实际的二进制库，比如 avr-libc 或其他由 avr-ar 与 avr-ranlib 创建的库，也可能是 Arduino 类型的库在链接之前编译时创建的目标文件。

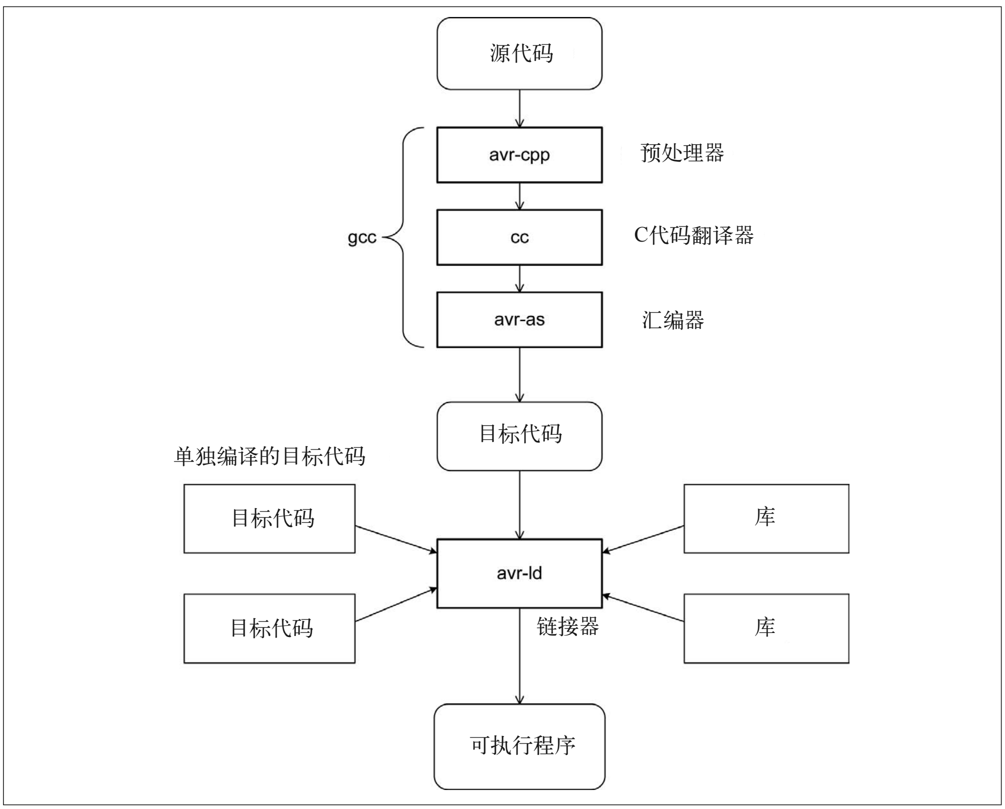


图 6-4：AVR-GCC 工具链

目前尚未提及的一个组件是 AVR-GCC 运行时支持库 avr-libc，它提供的许多函数都可以在标准的 C 运行时库（AVR-GCC）中找到。avr-libc 也提供了专门用于支持 AVR 微控制器的函数。6.2.5 节将详细讲解 avr-libc。

6.2.1　安装工具链

使用包管理器、Windows 安装程序、Mac OS X ZIP 文件安装 Arduino IDE 时，通常也会一起安装工具链组件。当然，如果不使用 Arduino IDE，也可以单独安装这些组件。

安装时，至少需要有如下软件包（Linux 系统下的包名称）：

- avr-lib
- avrdude
- avrdude-doc
- binutils-avr
- gcc-avr
- gdb-avr

如果使用的是 Windows 系统，那么可能还想安装其他 Unix/Linux 兼容工具，比如 grep、fgrep、ls、diff。这些工具已经安装在 Linux 与 Mac OS X 平台中。虽然此处不具体讲解，但你的确可以在运行 Solaris、BSD Unix（如 FreeBSD）或其他一些非常用的系统中安装 AVR GNU 工具链，只是需要你有正当的理由、极大的耐心，以及使用源代码包的大量技能。一般而言，使用已经带有现成可用安装包的系统会更简单一些。

1. 在Windows系统下安装

如前所述，Arduino 包同时带有必需的编译器与 binutils 程序。安装 Arduino IDE 时，程序将同时安装 IDE 所需的工具链组件。这些工具链组件被安装在 IDE 的主可执行代码所在的目录中。

Atmel 公司提供预编译的二进制安装包、基本文档、AVR gcc 源代码、binutils。你可以从 Atmel 官网（http://bit.ly/atmel-avr-win）获取这些安装包与其他资源。同时，官网也提供可用的源代码包（http://bit.ly/atmel-source）。

在 Windows 系统中安装 AVR-GCC 工具链的另一种简单的方法是，使用 WinAVR 包（http://winavr.sourceforge.net）。安装脚本创建一个目录（在我的机器上是 C:\WinAVR-20100110），并且把二进制可执行文件、库、包含文件放入其中。C:\WinAVR-20100110\doc 包含 PDF 与 HTML 两种格式的技术文档，方便用户阅读学习。请注意，自从 2010 年以来，WinAVR 就没有更新过，所以其中的一些工具链组件有点“陈旧”，但它们仍能在大部分项目中正常使用。看上去，Atmel 版本会更新一些。

可以在 GnuWin32 上找到大量针对 Windows 系统编译的 GNU 实用工具（http://gnuwin32.sourceforge.net）。GnuWin32 包含大量系统工具与实用程序，但其中并没有 gcc/g++ 编译器与 binutils 包。可以从 Atmel 或 WinAVR 下载位置获取。

借助 Google 搜索 Windows avr binutils，可以找到更多相关包与教程。如果喜欢使用 Eclipse 编辑器 /IDE 环境，并且尝试使用它们搭建 AVR 软件开发环境，请阅读 Protostack.com 中“Windows 系统下的 AVR Eclipse 环境”（http://bit.ly/protostack-eclipse）。

2. 在Linux系统下安装

大部分 Linux 系统中，安装 GNU gcc、binutils 以及其他 AVR 相关包非常简单，只要使用

软件包管理器选择相应包即可。可以同时安装“正常”的 GCC 工具链组件与 AVR 工具链组件，因为编译器与 binutils 工具的 AVR 版本都带有 avr- 前缀。我不建议你从源代码编译 avr-gcc 或工具链中的其他任何成员，除非你有令人信服的理由，并且擅长处理大量复杂的源代码包。

3. 在Mac OS X系统下安装

由于 Mac OS X 基于 BSD Unix 创建，所以许多 Linux 特性也同样适用于 Mac OS X。Adafruit 公司的“伙伴们”写了一份有用的指南（http://bit.ly/avr-osx），指导人们如何在 Mac OS X 环境中安装 AVR GNU 工具链。指向 Mac 包的页面链接已经失效，但你可以参考当前链接的下一段。

Object Development 的 CrossPack 开发环境中，包含在 Mac OS X 环境中开发 AVR 软件所需的所有工具链组件，并且不需要 Xcode 进行编译。可以从 Object Development 官方站点（http://bit.ly/crosspack-avr）下载。请注意，CrossPack 只包含工具链，并不提供 GUI IDE 或编辑器，所以需要你亲自动手安装。

对于 Mac OS X 系统，有相应版本的 Eclipse IDE 可用，并配备有 AVR 工具链。当然，Arduino IDE 也有适用于 Mac OS X 平台的版本可用。

6.2.2 make

对于小程序，直接在命令行中使用工具链程序即可。但随着程序规模变得越来越大，使用一些方法让这一过程自动化会更方便。在命令行环境中，可以选择 make 实现这一目的。

make 是一种解释程序，用于处理 makefile 文件。make 使用的语言不是通用的编程语言，而是一系列动作、规则、关系定义。你可以将其视为一种脚本，并且它经常被称为“宏语言”，因为其语句中会使用置换、替代来为其他工具构建命令。然而，make 能做的不止这些，它还能检测源文件的改动，并在不同源文件之间跟踪依赖关系（比如，如果 A 依赖于 B 与 C，那么当 B 发生变化时，B 就会被重新编译；A 也需要被重新编译，以便融入这些变化）。make 最初由 Richard Stallman、Roland McGrath、Paul D. Smith 创建。

许多用于大型代码开发的 IDE 都在后端使用 make 进行编译。PlatformIO 与 Ino 工具也使用 make，但它们会隐藏具体细节，并且事后进行清理。此外，还有一些工具自动为 make 工具创建输入。如果你使用 configure 工具构建过软件包，那么就会知道到这些工具的便利之处。

make 主要用于管理大量程序源文件。它能检测到哪些源文件发生了改变，并且当某些依赖文件发生改变时，确定哪些源文件需要重新进行编译。make 实用程序能够调用编译器、链接器、自动文档生成器，甚至其他 makefile 文件（在此情形下，源代码可能基于多个子文件夹发布）。当 avr-gcc 或 avr-ld 等工具遇到错误时，make 也能检测到。

了解如何使用 make 的一个好方法是，查看现有项目中的 makefile 文件。介绍 make 功能已经远远超出了本书的讨论范围（官方 GNU 手册超过 200 页），市面上有很多书详细讲解了 make 程序及其应用，各位可以自己阅读学习（推荐书目参见附录 D）。可以从 GNU 站点下载 PDF 格式的 make 官方用户手册（http://gnu.org/software/make/manual/make.pdf）。

6.2.3 avr-gcc

GCC 是 GNU Compiler Collection 的缩写。GCC 的基本思想是，针对不同语言使用不同的前端符号处理器，然后将产生的中间代码传递给针对特定目标平台的后端。avr-gcc 交叉编译器构建自 GCC 源代码，并进行了预先配置，使之能够专门为 AVR 系列的微控制器生成目标代码。GCC 能够为许多不同类型的处理器生成目标代码，其中包括 PC 中常用的 Intel CPU、SPARC RISC CPU、68000 系列大型微处理器、MSP430 MCU（Texas Instruments）、各种基于 ARM 的 MCU，以及其他许多处理器。

avr-gcc 与 avr-g++ 接收大量命令行参数，这些参数也被称为“开关”（switches），它们用来指定优化、编译模式、传递给链接器的参数、包含文件的路径、目标处理器。尽管有数百个命令行“开关”，但对于某种特定的目标处理器，只需要使用几个必要的“开关”就能成功编译代码。

编译通常涉及 4 个步骤：预处理、编译、汇编、链接。它们总是按照这个顺序进行。预处理器处理代码中的包含语句（`#include`），它将指定文件中的文本加载到当前源代码。通常，`#include` 语句只用来包含头文件（比如 stdio.h 或 stdlib.h），而非源文件（尽管使用 `#include` 语句包含源文件不是一个好主意，但我的确见过有人这样做）。预处理器也去除代码中的注释，并对代码中的条件预处理语句（比如 `#ifdef`、`#else`、`#endif`）进行解释。预处理器的输出是非常干净的源文件，它只包含“纯净”的代码，别无他物。你可以使用 `-E` 选项开关，使处理过程在预处理完成后立即停止，然后检查去除“杂质”之后的源代码。

本质上，GCC 是一个类 C 语言的“翻译器”，这些语言包括 ANSI C、C++、Objective-C 或 Objective-C++。从编译器输出的是中间汇编语言文件，它通常作为汇编器（avr-as）的输入，该汇编器之后会生成目标文件。当汇编器执行后，中间汇编语言文件就会被删除。可以使用 `-S` 开关让处理流程在运行汇编器之前停止，然后查看汇编语言输出。

`-c` 开关在不调用链接器的情形下创建一个目标文件。makefile 文件中会用到它，先编译所有源文件，而后在单个步骤中进行链接。编译器还有一些开关，用于优化、警告（打开它将生成许多警告信息）、路径说明（定位包含文件）。`-o` 开关用于指定被编译的可执行映像的名字，默认为 a.out。

关于 GCC 的更多信息，请阅读 GCC 手册页面。也可以从 GCC 在线文档站点（https://gcc.gnu.org/onlinedocs）下载 PDF、PostScript、HTML 格式的用户手册。GCC 不是一个简单的工具，它拥有大量可用选项。幸运的是，创建可执行代码时，你不必全部使用。

6.2.4 binutils

GNU binutils 是一系列程序，用于处理各种调用任务，将编译器的输出转换为处理器可执行的形式。表 6-2 列出了 Linux 系统下 binutils-avr 包中的内容。这套工具包含将二进制可执行文件变为适用于目标 AVR 微控制器运行的形式所需的一切，比如汇编、链接、转换等处理工具。在 GNUManuals Online 官方页面（http://bit.ly/gnu-manuals），可以找到 binutils 工具的使用手册（几乎所有其他 GNU 软件的使用手册都可以在该站点找到）。

表6-2：AVR binutils工具集

工具	描述
avr-addr2line	将地址转换为文件名、行号
avr-ar	生成目标代码文档、修改或提取代码
avr-as	AVR 的便携式 GNU 汇编器
avr-c++filt	对 C++ 符号进行解码
avr-ld	AVR 目标代码的 GNU 链接器
avr-nm	列出嵌入目标文件的符号
avr-objcopy	复制与翻译目标文件
avr-objdump	显示目标文件信息
avr-ranlib	为库文档生成索引
avr-readelf	显示 ELF 文件信息
avr-size	列出目标文件 section 的大小与总大小
avr-strings	打印二进制文件中的可打印字符串
avr-strip	从 AVR 目标文件中丢弃符号

为 AVR MCU 构建程序所必需的实用工具有 avr-ar、avr-as、avr-ld、avr-objcopy、avr-ranlib，而 binutils 套件中的其他组件对你开发软件可能有用，也可能无用。各个工具功能介绍如下。

avr-ar

avr-ar 用于创建二进制目标代码文档或者静态库，也可以对现存的库进行修改，或者从库中提取代码。二进制库文件（扩展名为 .a）由一系列二进制代码模块组成（如目标模块），并且带有主索引（由 avr-ranlib 创建，稍后讲解）。目标代码库也称为静态库，因为另一个程序中使用的任何组件都会被合并到最终可执行对象中，并成为其永久的或静态的一部分。而动态共享库则完全不同，通常 AVR 设备（或者其他任何微控制器）中不会使用它们，所以本书不会进行介绍。

avr-as

avr-as 是针对 AVR 系列 MCU 开发的便携式 GNU 汇编器。尽管它经常与 GCC 配合使用，但也可以将其用作独立的汇编器（6.4 节）。对于 AVR 微控制器，还有其他汇编器可用，我们也将在 6.4 节进行介绍。但是，只有 avr-as 才能与 gcc/g++ 编译器一起使用。

avr-ld

创建可执行的二进制文件过程中，avr-ld 通常是最后一步。链接器的主要功能是把两个或更多个目标文件组合起来，并在它们之间解析地址引用，重定位所需数据。

从多个目标文件构建可执行文件时，每个目标文件可能包含对某些函数或数据的引用，但这些函数与数据并不存在于当前目标文件，而存在于另一目标文件。libc 的 AVR 版本（接下来会讲到）就是这种情况的一个例子。比如，一个程序调用了 `atoi()` 函数（用于将 ASCII 码数据转换为整型数据），但本身并未包含 `atoi()` 函数的源代码。该程序被编译到一个目标文件（.o 文件）时，编译器将在调用 `atoi()` 函数的二进制代码中保留一个“洞穴”（hole）。链接器检测到这个空位置，在库（如 avr-libc.a）中找到

atio() 函数的地址，将该外部地址写入代码，然后把 atoi() 函数的目标代码包含到最终的二进制可执行映像中。

avr-objcopy

avr-objcopy 实用工具将目标文件的内容复制为另外一种格式，通常称为 IntelASCIIhex 格式文件，它非常适合用 Arduino bootloader 上传到 AVR MCU。avr-objcopy 也能生成名为 S-record 的 ASCII hex 文件，它一般用于 Motorola MCU（飞思卡尔）中。

Intel hex 文件拥有如下类似结构，请看下面代码，它显示的是 ATmega168 或 ATmega328 MCU 的 Arudino BootLoader 的开始行与结束行。

```
:107800000C94343C0C94513C0C94513C0C94513CE1
:107810000C94513C0C94513C0C94513C0C94513CB4
:107820000C94513C0C94513C0C94513C0C94513CA4
:107830000C94513C0C94513C0C94513C0C94513C94
:107840000C94513C0C94513C0C94513C0C94513C84
:107850000C94513C0C94513C0C94513C0C94513C74
:107860000C94513C0C94513C11241FBECFEFD8E036
<em>...更多数据...</em>
:107F700009F0C7CF103011F00296E5CF112480919F
:107F8000C00085FFB9CEBCCE8EE10E94C73CA2CD19
:0C7F900085E90E94C73C9ECDF894FFCF0D
:027F9C00800063
:040000030000780081
:00000001FF
```

为了简单起见，中间部分省略了许多行，但你可以在如下目录中找到源文件及其他相关内容。

/usr/share/arduino/hardware/arduino/bootloaders（Linux 系统）

C:\Program Files\Arduino\hardware\arduino\avr\bootloaders（Windows 系统）

如上所示，Intel hex 格式文件由多行记录组成，每一行记录由记录标志（冒号：)、记录长度（示例中除了最后 4 行外，全部为十六进制 10 或 16)、装载偏移（代码写入 MCU 闪存的位置，若有必要，BootLoader 可以对其进行修改）、记录类型、数据（以 ASCII 十六进制字符形式存在的实际代码，最多为 32 个字符，2 个字符 /B，共 16 B)、校验和。在维基百科上，你可以学到关于 Intel hex 文件格式的更多内容（https://en.wikipedia.org/wiki/Intel_HEX）——尽管几乎不需要直接查看 hex 文件。

可以使用 avr-objcopy 将二进制可执行文件转换为 hex 文件，如下所示：

```
avr-objcopy -O ihex execpgm execpgm.hex
```

与绝大多数 GNU 工具一样，objcopy 能做的远不止这些。它拥有大量命令行选项，其中大部分你可能永远不会用到。在 Sourceware.org 站点（http://bit.ly/swobjcopy），你可以找到关于 objcopy 的在线技术手册。

avr-ranlib

avr-ranlib 用于生成要包含在二进制目标文件中的索引。这将有助于加快链接过程，因为链接器会使用索引定位所需对象的地址，以便填充另一个目标文件中的链接“洞穴”。如果索引不可用，则链接器必须扫描整个库文件，逐个对象查找合适的匹配。

6.2.5 avr-libc

avr-libc 是 AVR 版本的 C/C++ 运行时库，它同 avr-gcc 与 avr-binutils 一起，构成针对 AVR 微控制器的 GNU 工具链的核心。

AVR 工具链提供的外部库（比如 avr-libc）位于标准位置下，在 Linux 系统中，对应于 /usr/lib/avr/lib/ 或 /usr/local/avr/lib，具体取决于 avr-gcc 是如何构建的，以及你的系统是如何配置的。其实，外部库可以位于任意文件夹下，但要保证链接器能够找到它们。

avr-libc 是一个关键组件，没有随 avr-gcc 与 binutils 一起提供。它是一个标准的 C/C++ 库，其中包含的函数都可以在标准的 C 库（比如 GNU libc）中找到对应的函数。只是受到某些限制，具体与 AVR MCU 的性能有关（比如容量有限的内存）。

表 6-3 列出了 avr-libc 提供的包含文件。如果你有过在“全尺寸”系统中使用 C 或 C++ 语言进行编程的经验，那么会熟悉其中的大部分包含文件。

表6-3：avr-libc提供的常见包含文件

文件名	描述
alloca.h	在调用者的栈帧中分配空间
assert.h	测试表达式为 false 结果
ctype.h	字符转换宏与 ctype 宏
errno.h	定义系统错误代码
inttypes.h	整型类型转换
math.h	基本数学函数
setjmp.h	定义非局部 goto 方法 `setjmp()` 与 `longjmp()`
stdint.h	定义标准整型类型
stdio.h	标准 I/O 设备
stdlib.h	通用工具
String.h	String 操作与实用工具

avr-libc 提供的一些包含文件只针对特定 AVR，它们在表 6-4 中列出。这些包含文件位于 /usr/lib/avr/include/avr 目录（就 Linux 系统而言）之下，其中一些为启动管理、时间延迟、EEPROM 访问、熔丝位设置、端口引脚功能定义了函数与常量，另一些则为特定类型的处理器定义了中断以及 I/O 映射。

表6-4： avr-libc提供的针对AVR的包含文件

boot.h	io90pwm316.h	iom169pa.h	iom32u2.h	iomx8.h	iotn45.h	iox128a3.h
builtins.h	io90pwm3b.h	iom169p.h	iom32u4.h	iomxx0_1.h	iotn461a.h	iox128d3.h
common.h	io90pwm81.h	iom16a.h	iom32u6.h	iomxx4.h	iotn461.h	iox16a4.h
cpufunc.h	io90pwmx.h	iom16.h	iom406.h	iomxxhva.h	iotn48.h	iox16d4.h
crc16.h	io90scr100.h	iom16hva2.h	iom48.h	iotn10.h	iotn4.h	iox192a3.h
delay.h	ioa6289.h	iom16hva.h	iom48p.h	iotn11.h	iotn5.h	iox192d3.h
eeprom.h	ioat94k.h	iom16hvb.h	iom640.h	iotn12.h	iotn84a.h	iox256a3b.h
fuse.h	iocan128.h	iom16hvbrevb.h	iom644.h	iotn13a.h	iotn84.h	iox256a3.h

（续）

interrupt.h	iocan32.h	iom16m1.h	iom644pa.h	iotn13.h	iotn85.h	iox256d3.h
io1200.h	iocan64.h	iom16u2.h	iom644p.h	iotn15.h	iotn861a.h	iox32a4.h
io2313.h	iocanxx.h	iom16u4.h	iom6450.h	iotn167.h	iotn861.h	iox32d4.h
io2323.h	io.h	iom2560.h	iom645.h	iotn20.h	iotn87.h	iox64a1.h
io2333.h	iom103.h	iom2561.h	iom6490.h	iotn22.h	iotn88.h	iox64a1u.h
io2343.h	iom1280.h	iom3000.h	iom649.h	iotn2313a.h	iotn9.h	iox64a3.h
io43u32x.h	iom1281.h	iom323.h	iom649p.h	iotn2313.h	iotnx4.h	iox64d3.h
io43u35x.h	iom1284p.h	iom324.h	iom64c1.h	iotn24a.h	iotnx5.h	lock.h
io4414.h	iom128.h	iom324pa.h	iom64.h	iotn24.h	iotnx61.h	parity.h
io4433.h	iom128rfa1.h	iom3250.h	iom64hve.h	iotn25.h	iousb1286.h	pgmspace.h
io4434.h	iom161.h	iom325.h	iom64m1.h	iotn261a.h	iousb1287.h	portpins.h
io76c711.h	iom162.h	iom328p.h	iom8515.h	iotn261.h	iousb162.h	power.h
io8515.h	iom163.h	iom3290.h	iom8535.h	iotn26.h	iousb646.h	sfr_defs.h
io8534.h	iom164.h	iom329.h	iom88.h	iotn28.h	iousb647.h	signal.h
io8535.h	iom165.h	iom32c1.h	iom88pa.h	iotn40.h	iousb82.h	signature.h
io86r401.h	iom165p.h	iom32.h	iom88p.h	iotn4313.h	iousbxx2.h	sleep.h
io90pwm1.h	iom168.h	iom32hvb.h	iom8.h	iotn43u.h	iousbxx6_7.h	version.h
io90pwm216.h	iom168p.h	iom32hvbrevb.h	iom8hva.h	iotn44a.h	iox128a1.h	wdt.h

avr-libc 也包含大量实用工具与兼容的包含文件，如表 6-5 所示。其实，avr/ 目录中的 delay.h、crc16.h、parity.h 文件指向 util/ 目录中的包含文件，所以当你在代码中包含 <avr/parity.h> 时，其实是在包含 <util/parity.h>。

表6-5：avr-libc提供的实用工具与兼容包含文件

文件名	描述
util/atomic.h	原子化与非原子化执行的代码块
util/crc16.h	CRC 计算
util/delay.h	忙 – 等延迟循环的便利函数
util/delay_basic.h	基本忙 – 等延迟循环
util/parity.h	生成奇偶检验位
util/setbaud.h	波特率计算辅助宏
util/twi.h	TWI 位掩码定义
compat/deprecated.h	废弃项目
compat/ina90.h	与 IAR EWB 3.x 兼容

有关使用 avr-libc 的内容，请参考用户手册。并且，请不要忘记检查包含文件自身，获取关于应用与限制的说明。请记住，在 AVR MCU 中使用 `malloc()`、`printf()` 等函数时，由于 AVR 内存有限，所以这些函数在使用上受到一定的限制。数学函数是另一个问题，因为 AVR MCU 不支持浮点运算，只支持整型与定点运算，所以必须借助对浮点处理器的模拟进行浮点运算。处理速度很慢，请尽可能不这样做。

avr-libc 的官方页面地址为 http://www.nongnu.org/avr-libc/。

> 可以在 http://bit.ly/avr-libc-manual 页面找到 PDF 用户文档的 bzip2 压缩版本。avr-libc 文档也涵盖 AVR 工具链。

6.3 从零开始构建C或C++程序

如果愿意，你当然可以在不使用 Arduino IDE、标准运行时 AVR-GCC、库的情形下构建自己的软件。不过，更容易的做法是，先使用 Arduino 工具快速实现并运行程序，然后再对程序中需要优化与定制的部分进行重写。

6.3.1 使用avr-gcc或avr-g++进行编译

通常不必了解关于 avr-gcc 或 avr-g++ 处理源代码的内部细节，但如果你愿意，当然可以从 GNU.org 站点读到关于 GCC 工具的更多信息（https://gcc.gnu.org/onlinedocs/gccint）。

事实上，gcc 与 g++ 命令不仅能编译源代码，还能做更多。当你使用如下命令时：

```
avr-gcc -mmcu avr5 -o test test_src.c -L../avrlibs –lruntime
```

gcc 将为 ATmega32U4 编译 test_src.c，调用 avr-ld 将其链接到位于 ../avrlibs 目录下的 libruntime.a 文件，然后将最终结果放入名为 test 的二进制可执行映像。

如果只想让编译器将代码编译到一个目标文件，以后再做链接操作，那么运行 avr-gcc 命令时可以使用 -c 开关（只做编译）。-c 选项常见于 makefile 文件中，makefile 文件先对多个源代码进行编译，然后再链接到一个可执行文件（也可能含有外部库），这是最后一步。对应的命令十分简单，如下所示。

```
avr-gcc -mmcu avr5 -c test_src.c
```

avr-gcc 也支持一系列用于优化、警告、替换包含文件（头文件）路径的开关。

6.3.2 多个源文件与make程序

有多个源文件时，make 程序是必不可少的。你可以编写自己的 makefile（我通常这样做），或者使用 arduino-mk 等工具。它是一个预定义的 makefile，包含使用 AVR-GCC 工具链组件构建 Arduino 程序的必需逻辑。

可以从 GitHub 下载 arduino-mk。某些 Linux 平台下有相应可用的可安装包。如果使用包管理器安装，那么包安装器很有可能会将主文件（arduino-mk）放入 Arduino 安装其组件的目录中。主网站是 Hardware Fun（http://bit.ly/hf-makefile），也可以从 GitHub（http://bit.ly/gh-makefile）下载 ZIP 文档。一旦安装好 arduino-mk，你可能还需另外安装一些支持功能，并且定义一些 shell 环境全局变量。请阅读 Hardware Fun 站点中的相关文档，以获取更多细节。

如果不熟悉 makefile 语言，那么可能需要面对一条陡峭的学习曲线。但如果想脱离 IDE，

深入了解嵌入式系统开发，我认为付出努力去学习是非常值得的。不是每个 MCU 都有可用的 IDE，也不是每个 IDE 都能完成在构建过程上进行直接控制时能够做到的所有事情。

我们可以重用示例 5-5 中的概念，创建一个简单的 makefile，如下所示。在本示例中，我们在 main.cpp 源文件中提供自己的 `main()` 函数，相当于 alarmloop.c 中的 `setup()` 与 `loop()` 函数、globals.c 中的全局变量，以及 defs.h 文件中的 `#define` 语句。makefile 文件内容如下：

```
CC = avr-gcc
LD = avr-ld
OC = avr-objcopy

SRCS   = main.c alarmloop.c globals.c
HDRS   = main.h globals.h defs.h
OBJS   = main.o alarmloop.o globals.o
EXEC   = alarm
TARGET = alarm.hex

$(TARGET): $(OBJS)
        $(LD) -o $(EXEC) $(OBJS)
        $(OC) -O ihex $(EXEC) $(TARGET)

main.o: main.c $(HDRS)
        $(CC) -mmcu avr5 -Wall -c main.c

alarmloop.o: alarmloop.c $(HDRS)
        $(CC) -mmcu avr5 -Wall -c alarmloop.c

globals.o: globals.c
        $(CC) -mmcu avr5 -Wall -c globals.c
```

该简单项目的目录中至少包含如下文件：

```
Makefile
main.c
main.h
alarmloop.c
alarmloop.h
globals.c
globals.h
```

输入 `make` 即可启动处理过程。make 文件位于当前目录下，拥有文件名为 Makefile（请注意，首字母 M 必须是大写）。如果找到，它将会被加载与处理。如果愿意，你可以为 makefile 使用另外一个名字，但你要告诉 make 需要找什么，如下所示：

```
make -f mymake
```

示例 makefile 中有几点值得简要介绍。首先是别名的声明，比如 CC、LD、SRCS。再次在文件中遇到它们时，它们就会展开，减少重复键入，并执行条件语句替换。

另一点要注意的是规则。示例中，我使用了显式规则，形式如下：

```
target: dependencies
  action
```

`action` 之前要添加制表符进行缩进。make 会将制表符用作语法的一部分，所以一定要注意不要使用空格符。此外，还要注意在一条规则下，你可以根据需要添加任意多个动作，但每个动作都要独占一行，并且行前要加一个制表符。

当 make 评估一个规则时，它会根据依赖关系确定需要什么来构建目标。就 main.o 而言，规则指定了 main.c 以及其他由 `$(HDRS)` 宏定义的包含文件（头文件）。如果目标尚不存在就会被创建，而如果存在，make 就会查看文件的时间与日期戳，以此判断是否有依赖关系发生改变。只要依赖发生改变，目标就会重新编译。从 `$(TARGET)` 规则看，任意源对象发生改变时，两个后续动作（链接与对象转换）一定会发生。

比如编辑 alarmloop.c 文件，对其进行了修改，然后运行 make，那么它首先会重新编译 alarmloop.c，生成新的 alarmloop.o，并再次构建 `$(TARGET)`。因为 alarmloop.o 已经发生改变。如果包含文件 alarmloop.h 发生改变，那么重建 `$(TARGET)` 之前，make 会重新编译 main.c 与 alarmloop.c。

创建 makefile 文件可能有些麻烦，但一旦你创建好它——除了添加新的源模块或者调整编译器与链接器开关设置之外——那么通常就不需要再经常调整了。

6.4 AVR汇编语言

如果你真想“榨干”AVR 设备的最后一点性能，那么应该选用汇编语言进行编程。汇编语言可不是谁都敢“玩”的，但它的确能够允许你完全控制 MCU 做什么以及何时做。在某些情况下，你可能需要使用这种级别的控制，以便从带有最少程序内存的设备上获得尽可能好的性能。对于一些 AVR MCU，比如 ATtiny 系列，汇编语言是唯一可用的编程语言。因为 C 或 C++ 程序经过编译后得到的文件可能太大，以至于无法装入这些设备的有限内存中。

本部分将对 AVR 汇编语言编程进行高度概括式的讲解。正如前言中所提到的，本书不是程序设计教程，而是一本硬件参考指南，帮助你获取更多相关细节信息。本部分旨在让你对汇编语言编程涉及的内容有大致的了解，以便帮助你判断它是否是你需要的。详细论述这一主题可能需要单独一本书，幸运的是，市面上已经有了这样的书。你也可以从各个网站下载大量有用信息，这些站点在 6.4.3 节中已经列出，请参考。

与其他任何语言相比，汇编语言更接近于底层硬件，因此有时也被称为“机器语言”。事实上，机器语言由各种指令的二进制代码组成，汇编语言则是机器语言的人类可读形式。汇编器的功能是，将代码从人类可读形式转换为机器可读形式，并且添加一些便捷特性，比如宏、符号名称、条件汇编，以及从外部库引用函数的能力。

在汇编语言中，我们能编写类似 `MOV R6,R7` 的语句，它将 R7 寄存器的内容复制到 R6 寄存器。该语句最终会被汇编器转换为二进制形式（机器语言），以供 MCU 执行。MCU 支持的各种操作都拥有相应名称，这些名称被称为“助记符”，紧接在助记符（若有）之后的参数称为“操作数”。

使用汇编语言编程是一门提供详细指令操作步骤的“艺术”，CPU 在这些指令的控制下做一些有用的事情。与此不同，C 语言等高级语言会帮助编程者处理具体细节，编程者不必自己决定要用哪些寄存器以及检查哪些状态位。使用汇编语言时，编程者与处理器的基础

逻辑之间没有任何中间层，即使最简单的操作也必须显式地进行描述。

6.4.1 AVR编程模型

从内部结构看，AVR MCU 由 AVR 核心、闪存、EEPROM 存储器与一系列周边电路组成。其中，AVR 核心包括指令寄存器与解码器、程序计数器、少量 RAM、各种状态位、32 个通用寄存器、算术逻辑单元（ALU）。图 6-5 描述了 AVR MCU 中的各种组件是如何组织在一起的。关于 AVR MCU 内部组成的更多细节，请阅读第 2 章和第 3 章的相关内容。

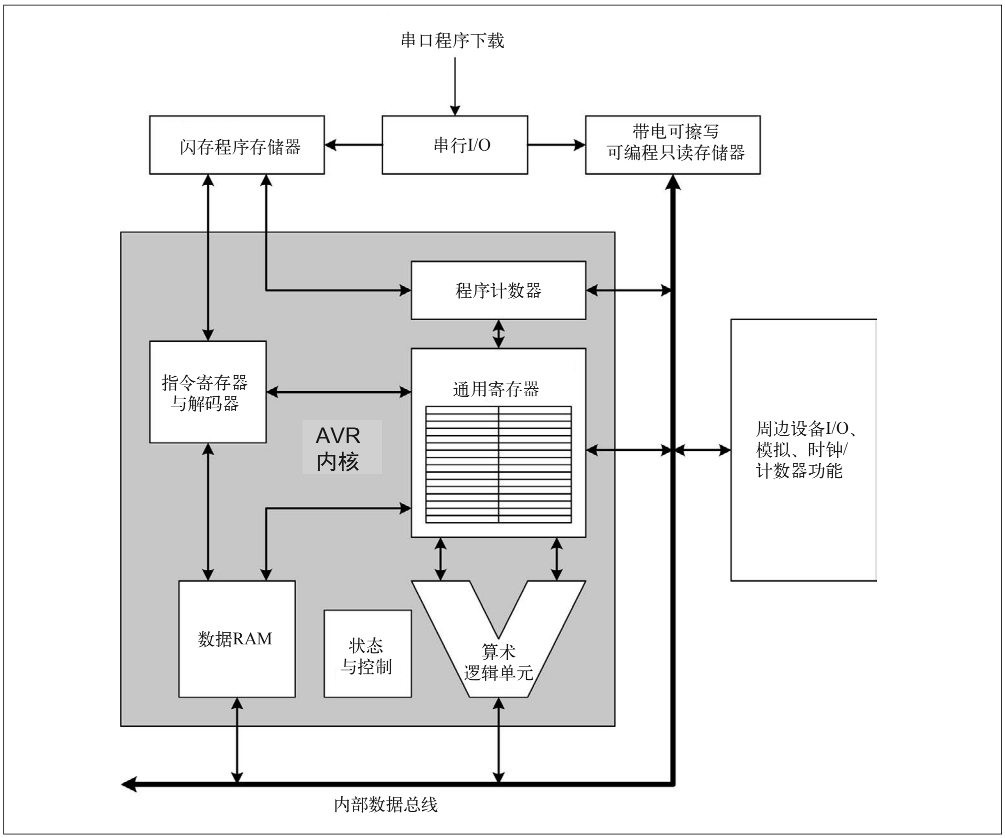


图 6-5：AVR CPU 框图

指令操作作用于寄存器与内存，将数据从一个寄存器复制到另一个寄存器，两个寄存器可以用来比较、交换数据，以及做加、减、乘、除运算（命名几个操作），能从闪存、EEPROM、RAM 中读取数据。周边电路也是借助寄存器进行控制与访问的，但它们并不是 32 个通用寄存器的核心集合的一部分。此处只介绍 MCU、内存、指令、寄存器的 3 个主要特征，描述如下。

内存组织

正如第 2 章所述，Atmel AVR MCU 采用哈佛架构。在哈佛架构中，程序代码存储于

只读存储器（闪存）中，可修改数据（变量）存储在独立的存储空间（AVR 核心中的 RAM）。其他微处理器有可能使用冯·诺依曼结构，在这种体系结构中，程序与数据共享相同的内存空间。

在 AVR MCU 中，通用寄存器和周围电路使用的 I/O 寄存器从技术上看都是可读可写内存空间的一部分。图 6-6 显示了 AVR MCU 中的内存是如何安排的。

图 6-6 特意描述的是一般情形。汇编伪指令 CSEG、DSEG、ESEG（由 Atmel 汇编器使用）分别表示代码、数据、EEPROM 内存空间。CSEG、DSEG、ESEG 内存空间的最高地址每种类型的 AVR 都不同。请注意，可选 BootLoader 的空间被保留在 CSEG 空间的末端。

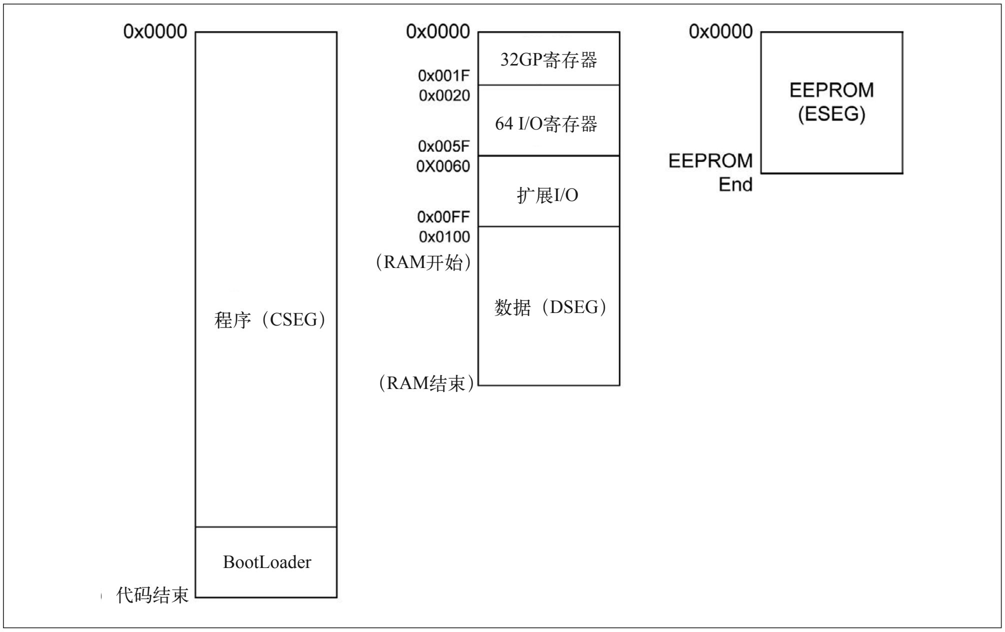


图 6-6：AVR 内存布局

指令处理

AVR MCU 采用单级管线（single-level pipeline）从闪存中获取、解码与执行指令。当一条指令正在执行时，下一条指令就会被从内存中预取出来。一旦当前指令执行完毕，预取出的指令即准备被解码执行。这一特征以及 AVR 核心的 RISC 性质允许 AVR MCU 在单个时钟周期内执行大多数指令。

寄存器

许多 AVR 指令在一个 8 位状态寄存器（SREG）中设置或清除状态位。每个位有特定的用途，如表 6-6 所示。某些位的真假状态（1 或 0）由 BREQ（相等则跳转）或 BRNE（不等则跳转）等后面跟着 CP 指令的指令负责检查，这些指令能够修改 Z、C、N、V、H、S 状态位。

32 个通用寄存器由 8 位寄存器组成，最后 3 对（26-27、28-29、30-31）可以用作 16 位变址（间接地址）寄存器，依次称为 X、Y、Z 指针。

少数指令能够对由两个 8 位寄存器构成的 16 位寄存器进行操作，其中编号较低的寄存器用于保存最低有效位。并且，最低有效位寄存器必须是偶数编号，所以 r0:r1 合法，而 r1:r2 不合法。

表6-6：AVR SREG状态位

位	符号	功能
0	C	进位标志
1	Z	零标志
2	N	负标志
3	V	补码溢出标志
4	S	符号标志位
5	H	半进位标志位
6	T	BLD 与 BST 指令所用的传送位
7	I	全局中断触发 / 禁用标志位

除了状态寄存器、通用寄存器、I/O 寄存器之外，AVR 还有一个程序计数器（PC）寄存器与一个栈指针（SP）寄存器。它们能够被某些指令（比如跳转指令会修改 PC）或子程序调用指令（同时使用 PC 与 SP 寄存器）修改。比如，遇到 `CALL` 指令时，PC 的当前值就会被修改为 `CALL` 的下一条指令，然后将其压入栈；再将 PC 修改为子程序的起始地址，执行子程序遇到 `RET` 指令返回时，从栈弹出之前保存的地址，然后从 `CALL` 指令的下一条指令继续执行程序。

当然，关于 AVR MCU 与 AVR 汇编语言（此处不介绍）的内部工作机制还有更多细节。如果想了解关于 AVR 汇编语言编程的更多内容，强烈推荐阅读一两本相关书籍。附录 D 包含了一些相关建议，其中提到的链接也非常有用。

6.4.2　创建AVR汇编语言程序

在不同汇编器之下，单行注释可能以分号（;）开头，或者以井号（#）开头，也有可能以其他字符开头。GNU 汇编器（avr-as）支持多行注释，这些注释被放入 /* 与 */ 之间，这种注释可以在 C 代码中经常看到。相关细节请参考汇编器的相关技术文档。

汇编语言中的注释非常重要。比如你有如下汇编语言代码：

```
LOOP:
LDI     R16, 0X00
OUT     PORTB, R16
LDI     R16, 0XFF
OUT      PORTB, R16
RJMP  LOOP
```

这段代码的功能是什么呢？其实这段示例代码的功能很简单，先将 0 或 255（0xFF）载入 R16 寄存器，然后再将 R16 寄存器中的内容写入 PORTB。PORTB 引脚将依据 MCU 执行

循环的速度进行打开和关闭操作。

在此基础之上可以进一步改进代码，添加其他功能，比如程序起始地址、端口初始化等。改善之后的代码如下所示：

```
; 设置power-up/reset向量
.ORG 0X0000
RJMP MAIN

; 代码入口
.ORG 0X0100
MAIN:
LDI  R16, 0XFF ; 把0b11111111加载到R16
OUT  DDRB, R16 ; 设置端口B数据定向寄存器

; 无限循环 - 切换端口B引脚为on与off

LOOP:
LDI            R16, 0X00  ; 把0装载到R16
OUT            PORTB, R16 ; 写到端口B
LDI            R16, 0XFF  ; 把0xFF装载到R16
OUT            PORTB, R16 ; 写到端口B
RJMP  LOOP       ; 跳回重做
```

上面代码中，我添加的注释有些多，但在汇编语言中几乎每行都能看到注释并不鲜见。请记住，在汇编语言中，即使编写类似 Hello world 的简单程序，也需要使用比 C 或 C++ 语句多得多的语句，并且这些汇编语句可能也不会很直观。

大多数汇编器支持一系列预定义的关键字，称为“伪指令”。上面示例中有一个 `.ORG` 伪指令。表 6-7 列出了一些常用的伪指令。这些关键字之所以被称为“伪指令”是因为，它们会指示汇编器产生特定关联或执行某些操作。

表6-7：AVR汇编器伪指令

伪指令	操作
BYTE	保存 1 字节或多字节作为变量
CSEG	使用代码段
CSEGSIZE	配置代码段内存大小
DB	定义 1 常量字节或多字节
DEF	为寄存器定义一个符号名
DEVICE	定义为哪个设备进行汇编（目标）
DSEG	使用数据段
DW	定义单个或多个常量字（16 位值）
ENDM, ENDMACRO	宏定义结束
EQU	将一个符号指派给表达式
ESEG	使用 EEPROM 段
EXIT	从文件退出
INCLUDE	从另一个读取并包含源文件

（续）

伪指令	操作
LIST	启用列表文件生成
LISTMAC	启用列表文件中的宏扩展
MACRO	宏定义开始
NOLIST	禁用列表文件生成
ORG	设置程序起始地址
SET	将符号指派给表达式

下面代码片段描述了一些伪指令是如何使用的。

```
; 当包含外部文件时禁用清单生成
; 这有助于清单整洁又不凌乱

.NOLIST
.INCLUDE "macrodefs.inc"
.INCLUDE "mcutype.inc"
.LIST

; 定义宏实现某个功能

.MACRO SUBI16           ; 定义宏
       SUBI @1,low(@0)  ; 减低字节
       SBCI @2,high(@0) ; 减高字节
.ENDMACRO                    ; 宏定义结束

SUBI16 0x2200,R16,R17  ; 从R17:R16减去0x2200
```

6.4.3 AVR汇编语言资源

如果想深入了解 AVR 汇编相关内容，可以从网上找到许多有用的信息与教程等资源。

- AVR 汇编器网站（http://avr-asm.tripod.com）拥有很多有价值的信息，各种信息分门别类、整洁有序。
- AVRbeginners.net（http://www.avrbeginners.net）是一个非常棒的网站，讲解了许多有关 AVR MCU 内部工作原理的细节，也提供了一些汇编语言示例。
- Atmel 提供 AVR 汇编器在线参考文档（http://bit.ly/avr-assembler），在 AVR Assembler User Guide（http://bit.ly/avr-assembler-guider）中，也能够看到关于 Atmel 汇编器的描述。虽然与 avr-as 汇编器不一样，但一般原则都是类似的。
- 在 GNU Manuals Online（http://bit.ly/gnu-manuals）文档中，可以找到有关 avr-as 汇编器的描述，avr-libc 文档中也有一个概述。
- Gerhard Schmidt 制作了一个网站（http://bit.ly/avr-overview），包含许多有用信息，并且支持英文与德文两种语言。其中也提供 PDF 格式的文件（http://bit.ly/avr-overview-pdf），供用户阅读学习。
- 可以从约翰斯·霍普金斯大学网站（http://bit.ly/jhu-assembler）下载汇编语言概要。

可以说，这只是冰山一角。请参考附录 D 以获取推荐书目。

6.5 上传AVR可执行代码

将代码编译或汇编到一个可执行文件只走完了一半路程，另一半则要将可执行代码上传到 AVR MCU 以便执行。有多种方法可以完成这一步，包括使用 Arduino BootLoader、许多 Arduino 开发板上的 ICSP 接口（至少相关针脚通常是可用的）、JTAG 接口。

6.5.1 系统内编程

Atmel 应用指南 AVR910（http://bit.ly/avr-insystem）描述了 AVR MCU 上使用的系统内编程接口。它在 Arduino 开发板上称为 ICSP 接口，本质上是对 AVR MCU 的 SPI 接口的扩展。图 6-5 中，serial I/O 显示为独立的功能模块，因为它能直接与 MCU 中的闪存、EEPROM 存储器进行通信。

> **闪存使用寿命**
>
> 所有闪存能承受的写操作次数是有限的。在现代组件中，这通常是一个非常大的数字（不同部件是不同的），写入周期（write cycles）约为 10 000 或者更多。与桌面电脑常见磁盘驱动器的工作次数相比，听起来并不是很多。但请记住，即使它能被看作一种慢速磁盘驱动器（比如 USB），但微控制器中的闪存不是磁盘驱动器。它被用于加载程序以便运行，运作期间不会存储运行时 AVR-GCC 数据（微控制器通常有一些可用的 RAM 用作此用途，像 microSD 晶片，预计约有 100 000 写入周期，它们能用于记录大量运行时 AVR-GCC 数据）。这样算来，如果每天一次使用程序代码重载闪存内容，可用 4~5 年。

图 6-7 展示了 Arduino Uno（R2）开发板上主 ICSP 连接器的引脚。此外，该开发板上还有一个 ICSP 连接器，用作控制 USB 接口的 AVR MCU。它拥有相同的连线，但真的没有理由使用它，除非需要对那块 MCU 重新编程（这可能会失去 USB 功能）。请注意，Atmel 也定义了一种 10-pin 的连接器，但大部分 Arduino 类型的开发板都未使用它。10-pin 连接器与 6-pin 连接器拥有相同的信号，更多的接地连接将占用多出的 4 个引脚。

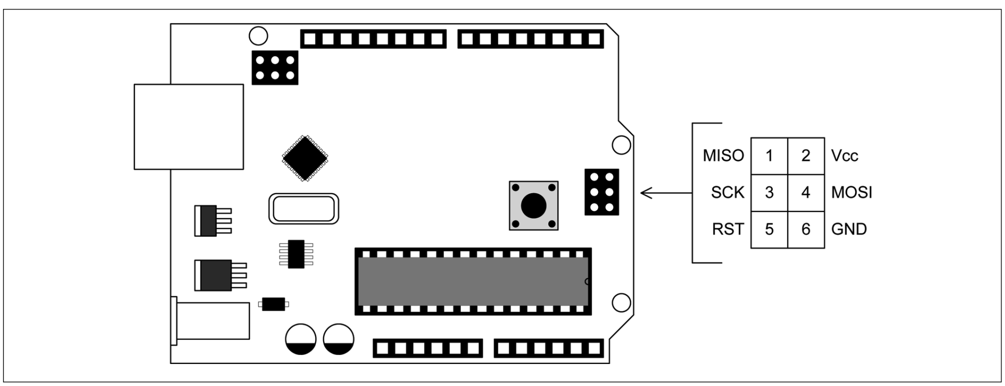


图 6-7：Arduino Uno R2 上的 ICSP 连接器

6.2.4 节介绍了 avr-objcopy 工具，重点放在如何将可执行的二进制映像转换为 ASCII hex 文件。这样做是因为 Intel hex 文件可以用于定义记录段、地址、文件结尾，然后通过通信连接发送数据。该过程中，发送纯二进制数据可能引起问题。AVR MCU 使用二进制数据对闪存编程，但在 AVR910 应用指南中，你会发现无论如何编程都会对整个过程有很大的控制。hex 文件是提供给编程设备而非 AVR MCU 使用的。

6.5.2 使用Bootloader编程

就 Arduino 开发板而言，编程设备就是闪存中的 Bootloader。它允许 MCU 为自己编程，但是这需要额外付出一些闪存空间以存放 Bootloader。若没有 Bootloader，编程设备自身要处理最终数据的位置（目标地址）、MCU 配置位（熔丝位），以及其他底层细节（关于 AVR MCU 熔丝位的更多细节，请参考 3.4 节）。

一定要注意，Arduino Bootloader 固件使用 AVR 串口引脚（RxD 与 TxD 依次对应于 D0 与 D1），所以如果想使用类似 RS-232 的转换器，可以用它为 AVR 编程，或者使用 USB-to-serial 转接器，如图 6-8 所示的 SparkFun 模块。在一些 Arduino 开发板（使用 FTDI FT232L、ATmega8 或 ATmega16U2 作为 USB 接口）上，电路图显示接口芯片或 MCU 通过 1K 电阻使用 D0 与 D1 串行引脚，DTR 信号为主 AVR MCU 产生复位。

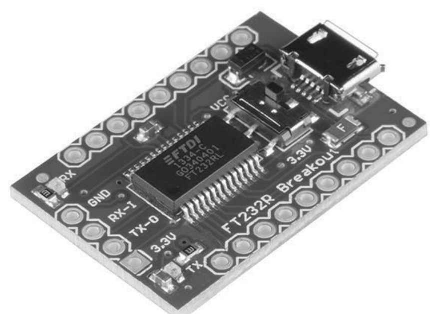


图 6-8：SparkFun USB-to-serial 转接器

你仍然可以使用 D0 与 D1 引脚，只要能把它们正确地进行隔离。图 6-9 的电路显示，在 Arduino 中一般如何将 FTDI FT232L 连接至 AVR MCU。请注意，这并不适用于 Leonardo 与其他使用 ATmega32U4 的开发板，它带有内置的 USB 接口。

6.5.3 不使用Bootloader上传

如果你编写的程序真的需要最大限度使用 AVR 闪存空间，或者你不想使用标准 Arduino 方式中的 D0 与 D1 引脚上传程序，那么可以直接使用 ICSP 接口加载已经编译好的程序。这种方法适用于直接来自 Atmel 的“纯净”的 ATmega MCU，当然也适用于 Arduino 开发板。

不使用 Arduino IDE 的情况下，上传程序需要直接与 AVRDUDE 实用程序（6.5.5 节）进行交互。通常这也需要使用 makefile 与其他一些技术产生编译代码。

如果你想享受 Arduino IDE 的便利之处，让它为你处理编译事务，那么直接使用 Arduino IDE 即可。Arduino IDE 支持直接对 AVR 设备编程，允许你通过 Tools 主菜单中的子菜单项选择编程设备。在 File 下拉菜单中，选择“使用编程器上传”即可启动。如果你重写了 Bootloader 固件，并决定以传统方式重新使用 Bootloader，那么就需要按照 6.5.8 节的讲解重新进行加载。

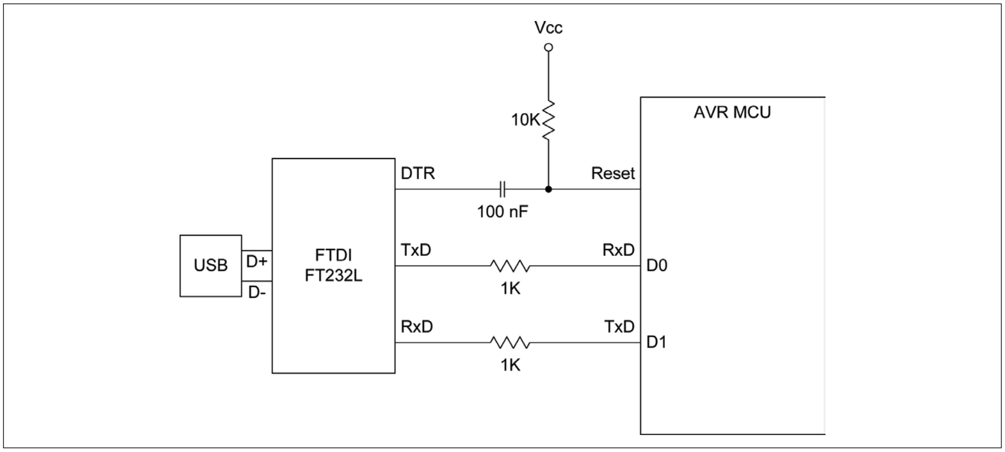


图 6-9：使用 FTDI 转换器 IC 的 Arduino USB 接口

当然，如果内存空间不是问题，并且已经安装 Arduino Bootloader，那就不必移除 Bootloader。无论有无 Bootloader，ICSP 接口都能正常工作。如果选择使用 ICSP 接口，则可以完全忽略 Bootloader。

如果不使用 Bootloader，那么必须使用特殊的编程设备才能进行编程。Atmel 提供了 AVRISP MKII、Atmel-ICE 等工具，它们都很典型，功能强大，兼容性高。令人遗憾的是，为了支持 Atmel-ICE，Atmel 终止了 AVRISP MKII，但现在还有很多兼容设备可用。请注意，AVRISP MKII 并不支持 JTAG。

由于 ISP 接口本质上是 SPI 串口，所以你能使用各种设备完成这项工作，包括另一类 Arduino 开发板（更多内容稍后讲解）。一些容易获得的编程设备有 USBtinyISP（Adafruit，图 6-10）和 Pocket AVR Programmer（SparkFun，图 6-11）。

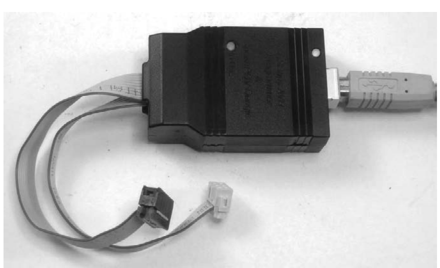


图 6-10：来自 Adafruit 的 USBtinyISP（已组装）

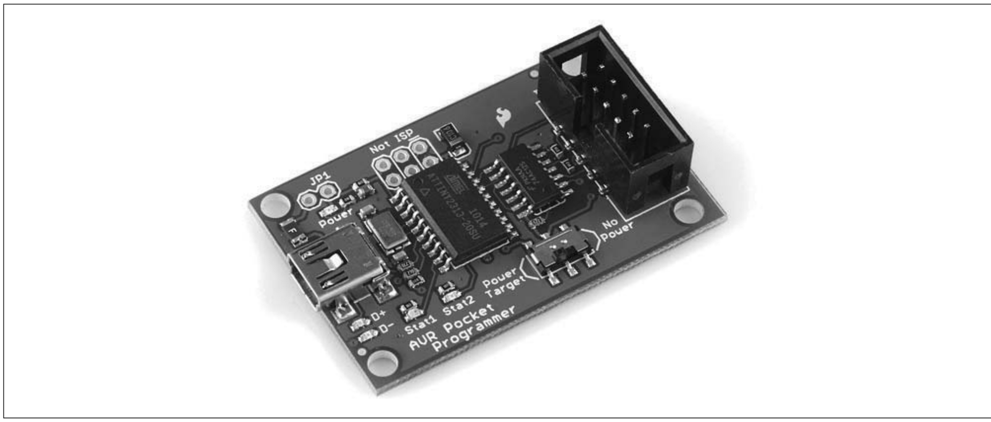


图 6-11：来自 SparkFun 的 Pocket AVR 编程器

USBtinyISP 编程器是一个套件，但组装相对容易。在 Adafruit 网站上可以学到更多相关内容（http://bit.ly/usbtinyisp-avr）。Pocket AVR Programmer 是预先装配好的，可以在 SparkFun 网站（http://bit.ly/sf-pocketavr）找到更多相关信息。

除了将软件加载到 AVR 之外，编程器还可以用于读取寄存器、检查内存、设置熔丝位。获取熔丝位设置功能是购买 ISP 设备的重大理由。有关熔丝位的更多信息，请阅读 3.4 节。

6.5.4 JTAG

JTAG 是 Joint Test Action Group 的缩写，它是一个底层接口，用于访问集成在 MCU 或其他逻辑设备中的调试功能。可以在 IEEE 文档 Standard Test Access Port and Boundary-Scan Architecture（IEEE Standard 1149.1-1990）中找到正式定义。从 IEEE 站点（https://standards.ieee.org）可以获取最新版本以及其他相关标准。

请注意，并非所有 AVR MCU 都支持 JTAG。由 Atmel 选择指南与数据手册可知，目前 XMEGA 系列设备支持它，但 8 位 Mega 系列部件（像 Arduino 开发板中使用的那些）不支持。然而，由于目前有很多种 AVR MCU 可用，所以有些 XMEGA 部件不支持 JTAG，而一些 Mega 部件则支持。这种情况是完全有可能的。

其实你通常并不需要 JTAG 的高级特性。想借助调试器逐步跟踪代码时，或者在运行中检查寄存器的内容时，使用 JTAG 接口非常方便。但更常见的情况是，只要使用示波器或逻辑分析器查看引脚即可获取大量有用信息。

至于访问 AVR 的内部功能，你可以使用类似 USBtinyISP 的编程器设置内部 AVR 熔丝位，或加载 EEPROM。因此，除非真的需要使用 JTAG 工具，否则完全可以省下这笔费用。

6.5.5 AVRDUDE

你最好有 AVR 编程器，但它也需要有“东西”为其提供要上传到 AVR MCU 的数据。这就是 AVRDUDE。

AVRDUDE 是 AVR Download UploaDEr 的缩写，它是一个实用程序，用于从 AVR MCU 的内存空间上传和下载代码与数据。图 6-12 显示了运行不带任何参数的 avrdude 命令时输出的内容。

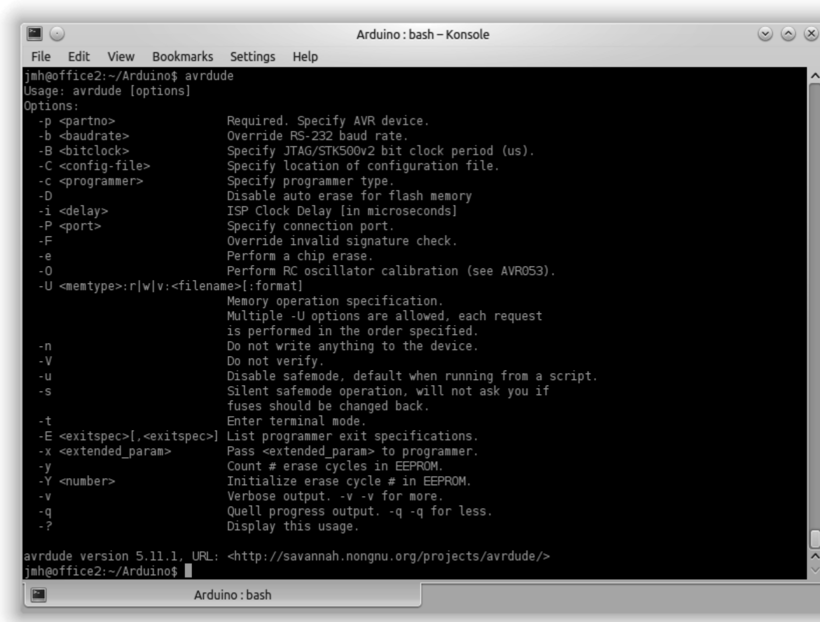


图 6-12：AVRDUDE 帮助输出

AVRDUDE 也可以用于对板载 EEPROM 内存、熔丝位、锁定位进行编程。它既可以运行在命令行之下，也可以运行在交互模式下。将 AVRDUDE 集成到一个脚本或 makefile 时，命令行模式很有用；而交互模式能在 MCU 内存中进行搜索，在 EEPROM 中修改个别字节，或者对熔丝位、锁定位进行操作。

AVRDUDE 支持各种编程设备，包括 Atmel STK500、AVRISP、AVRISP MKII、串行位拆裂（serial bit-bangers）、并行端口接口。更多相关内容可以查看 PDF 格式的使用手册（http://bit.ly/avrdude-pdf）。

Arduino IDE 使用 AVRDUDE 处理上传过程，通过从首选项对话框启用上传输出，可以看到命令行的样子。将第 5 章中的简单入侵报警程序上传，将看到如下输出（请注意，我对输出结果做了换行处理，以使它们更适合页面大小）：

```
/usr/share/arduino/hardware/tools/avrdude
-C/usr/share/arduino/hardware/tools/avrdude.conf
-v -v -v -v -patmega328p -carduino -P/dev/ttyUSB0 -b57600 -D
-Uflash:w:/tmp/build2510643905912671503.tmp/simple_alarm.cpp.hex:i
```

请注意，-p、-c、-P、-D、-U 是非常重要的命令选项。一些常用的命令选项如表 6-8 所示。多个 -v 选项只用来让 AVRDUDE 尽可能详细地输出内容。

表6-8：AVRDUDE命令行开关（选项）

开关	函数	功能
`-p`	Processor ID	识别连接到编程器的部件。在我们的例子中，被编程的 Arduino 开发板的确有 ATmega328p，它是 Duemilanove。
`-c`	Programmer ID	指定要使用的编程器。示例中，编程器是 Arduino bootloader
`-C`	Configuration	指定要使用的配置文件
`-P`	Port name	识别编程器连接的端口。在 Linux 系统下，使用位于 /dev/ttyUSB 的虚拟串行端口
`-b`	Port baud rate	重写默认波特率
`-D`	Auto-erase	关闭闪存自动擦除
`-U`	Upload	上传规格说明

请注意，命令行中的 `-U` 开关由多个部分组成，它定义了内存目标（flash）、模式（写），以及包含 hex 格式可执行文件的源文件。位于参数字符串最后的 `i` 指定 hex 源文件是 Intel 格式。

6.5.6 将Arduino用作ISP

将实用程序加载到 Arduino，可以将其变为另外一个 Arduino 的 ISP。借助这项技术，你可以上传一个新的 Bootloader。对此，Arduino 网站给出了简要说明（http://bit.ly/arduino-isp）。

基本上，需要的全部就是 Arduino IDE 提供的程序与两个 Arduino 开发板。图 6-13 显示两个开发板是如何连接在一起的。图中 Duemilanove 被用作 Uno R3 的 ISP 设备。它也能与没有相同 SPI 信号引脚的开发板一起工作，比如 Leonardo，但你需要为此做些调整。更多细节请参考 Arduino 文档。

6.5.7 Bootloader运作

Bootloader 用于从主机开发系统为 AVR 接收程序。安装在 Arduino 开发板上的微控制器预装了 Bootloader，对于大多数应用，几乎不需要移除或重载 Bootloader 软件。除非你有令人信服的理由，必须收回 Bootloader 占用的内存，否则最简单的方法是不要碰它，并充分利用它带来的好处。

Arduino Bootloader 的运作方式类似于其他任何带有闪存的微控制器，其主要目标是将用户提供的程序放入板载内存，然后将控制转交给新程序以便执行。当 Arduino 通电时，它会先执行 Bootloader。然后，Bootloader 检测 USB 接口是否有新输入的程序数据。一段时间后，若没有检测到任何数据，就会执行已加载到闪存主区的程序。

在新版本的 Arduino 中，针对输入数据的检测甚至在 AVR 处理器运行以前存储的程序的过程中也有发生。检测到有新程序上传时，现有程序代码执行被中断，Bootloader 全面接管处理器。在较旧的 Arduino 开发板上，只要 IDE 开始上传编译好的程序，复位按钮（Reset）必须被按下，以便检测上传。你可能需要这样重复几次才能把握好时机。请注意，一些 Arduino 兼容开发板也采用这种方式。

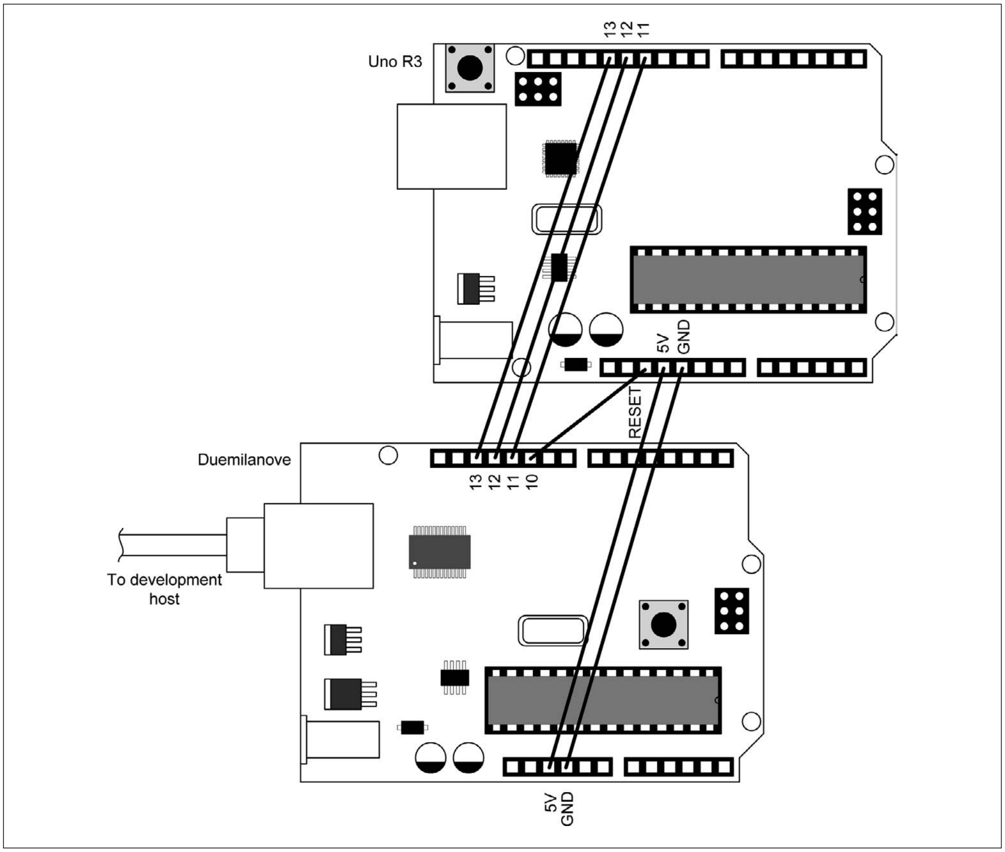


图 6-13：使用一个 Arduino 作为另一个 Arduino 的编程器

一旦 Bootloader 判断上传输入的数据是一个合法程序，它就会解锁板载闪存，开始读入新程序，并将其写入闪存（不是 Bootloader 自身驻留的位置）。一旦上传完成，内存将再次被锁定，以防发生意外修改。还会产生一个中断，将处理器重定向（有时称为“引向”）到新程序的起始地址处。

当前版本的 Arduino Bootloader 能以 19 200 波特率（19.2 千波特）读数据，或者约 1900 B/s。旧版本的 Bootloader 以 9600 波特率侦听输入数据，如果有必要，你可以更改 Arduino IDE 的行为以便适应。即便采用 19.2 K 波特率，将一个大程序传送到目标处理器也需要花费一些时间。

在 Arduino.cc 网站可以找到 Bootloader 源代码。如果真想了解 Bootloader 如何工作，那么阅读其源代码是非常值得的。认真阅读后就会发现，操纵闪存部分的代码是由 AVR 汇编语言编写的。从这一点看，Bootloader 代码更接近于硬件。与微控制器的内部控制寄存器进行交互，汇编语言是完成这一任务的最佳语言。Bootloader 源代码也能帮助我们了解微控制器的底层工作原理，当一个程序被加载并运行时，其背后发生了许多事情。

6.5.8　更换Bootloader

如果你需要（或希望）在 Arduino 的 AVR 处理器上安装新的 Bootloader，那么应当使用图 6-10 与图 6-11 中列出的设备，直接通过 ICSP 端口为微控制器的闪存进行编程；或者也可使用 Arduino-to-Arduino“把戏”，6.5.6 节已经讲过。

出于这一目的，Bootloader 驻留在 AVR 闪存的一段特定区域中，该区域大小为 256 B~4 kB，具体取决于 AVR 类型。Arduino IDE 支持上传 Bootloader，但需要用到前面提到的那些编程设备。在 IDE 的 Tools 菜单下选择你使用的编程器类型，编译 Bootloader，然后将其上传到 AVR 微控制器。更多细节请参考 Arduino 内置的帮助内容，或者访问 Arduino 官方网站。

6.6　小结

本章简单介绍了多个主题，涵盖从 AVR 工具链组件到 make 实用程序、汇编语言编程、AVR Bootloader 等内容。即使你从未用过这些工具或技术，但它们仍然有助于你了解 Arduino IDE 背后的工作机制。本章也可以帮你了解在 Arduino IDE 出现之前，嵌入式系统开发中所用的各种工具与技术。

如果你想探索本章所列主题的更多细节，一定要好好利用提供的各种参考。不要忘记查看附录 D，从中可以获得更多信息资源。嵌入式微控制器设备是现代文明的重要组成部分，在你能看到的每台计算机以及看不到的地方——比如电视遥控器、微波炉、音响、DVD 播放器、交通信号控制、汽车、你桌面上的电脑键盘——都有它们的身影。所有这些小设备都使用本章介绍的技术进行一次性编程，时至今日，还有很多设备仍然在采用这些方法。

第 7 章

Arduino库

Arduino IDE 自身带有一系列函数库，方便用户在程序中使用。这些库包含的函数用于访问周边设备，比如以太网接口、液晶显示器、传统串口，以及其他外围设备。

请注意，尽管“库”（Library）这一术语用来描述“辅助代码”（auxiliary code），但是从预编译模块的意义上说，模块本身并非总是人们口中所说的“库”，比如 Unix 或 Linux 中的 .a（存档文件）或 .so 库（共享对象）。在许多情况下，它们只是标准的 C 或 C++ 源文件（当然，带有 AVR-GCC 限制），但最终结果大多一样。必要时，库代码会随程序代码一起被编译到对象文件中，并与程序进行链接（第 5 章与第 6 章）。在其他情况下，库其实是一个二进制对象，比如 avr-libc 库套件提供的各种组件。如果你想了解库或外部代码模块从何而来，请查看 avr-libc 以及 Arduino 文档。

当程序与库模块被编译之后，链接器工具就会在库组件与用户提供的函数之间解析地址引用，然后将所有组件绑定到一个可执行的二进制映像。Arduino IDE 使用 AVRDUDE 实用程序（第 6 章）处理与板载 Bootloader 的交互过程（第 5 章），将编译好的二进制代码传送到 AVR 设备，并将其存储到处理器的板载内存。

有关使用 Arduino IDE 开发软件的内容，请参考第 5 章中的相关介绍。第 6 章只讲解了如何使用 AVR-GCC 工具链进行代码开发。

7.1 库组件

Arduino 开发环境带有一系列支持库，用于支持基于 USB 连接的串行 I/O、EEPROM 读 / 写、以太网 I/O、LCD 接口，还支持舵机与步进电机驱动器。接下来讲解这些库组件。

这些讲解都是简明扼要的。更多细节与使用示例请参考 Arduino 库页面（http://bit.ly/arduino-libraries），或查看 Arduino IDE 中的 HTML 格式参考文档（内置于 help 菜单）。请记住，Arduino 网站中有最新版本库的最新文档，而 Arduino IDE 中的文档只包含对 IDE 发布时的库的说明。

在 ArduinoIDE 工具栏中，依次选择“项目 -Import Library”菜单，即可查看可用的库列表。在该菜单之下，你也可以看到自己添加到 IDE 中的各个库（有关向 IDE 添加库的内容，请参考第 5 章）。

下面讲解的内容只涵盖安装好 Arduino IDE 之后所提供的基本库（从 Arduino 与其他扩展板提供商那里可以找到更多可用的函数库）。

EEPROM
该库用于支持对 AVR 内置 EEPROM 的读写操作，并可实现永久性存储。

Ethernet
与 Arduino 以太网扩展板一起使用，实现以太网连接。

Firmata
借助标准的串行协议，为 Arduino 与主机应用程序之间的通信提供支持。

GSM
与 GSM 扩展板配合使用，以便连接到 GSM/GPRS 网络。

LiquidCrystal
该库中的函数用于控制液晶显示（LCD）。

SD
提供对读写 SD 闪存卡的支持。

Servo
该库中的函数用于控制伺服发动机。

SPI
支持使用串行外设接口（SPI）总线。

SoftwareSerial
在任意数字引脚上实现串行通信。

Stepper
包含一系列用于控制步进电机的函数。

TFT
该库中的函数可以在 Arduino TFT 屏幕上绘制文字、图像、形状。

Wi-Fi
提供对 Arduino Wi-Fi 扩展板的支持，实现无线网络。

Wire

支持双线接口（TWI/I2C），在设备或传感器网络上收发数据。

Esplora

该库（仅适用于 Esplora）中的函数用于访问装配到 Esplora 开发板上的各种驱动器与传感器。

USB

适用于 Leonardo、Micro、Due、Esplora 开发板，用于实现基于 USB 连接的串行 I/O。

Keyboard

发送击键到所连接的计算机。

Mouse

控制所连计算机上的光标移动。

7.1.1 EEPROM

EEPROM 库用于支持对 AVR 内置 EEPROM 的读写操作，并可实现永久性存储。EEPROM 是永久性存储器，开发板断电后，它仍然能够保存之前写入的数据。虽然微控制器的主闪存也是非易失性的，但新的可执行代码上传到开发板时，EEPROM 不会受到干扰。确切地说，它必须通过软件进行访问。各种 Arduino 开发板上使用的不同类型的 AVR 微控制器拥有不同数量的 EEPROM 存储空间，从 ATmega168 的 512 B 到 ATmega1280 与 ATmega2560 的 4 kB。有关微控制器的更多细节，请参考第 3 章相关内容。

`EEPROM` 类定义了一些函数，用于读、写、更新 AVR 板载 EEPROM 的内容。使用任何 EEPROM 函数之前，必须先对 EEPROM 类进行实例化。

库包含文件（EEPROM.h）将 `EEPROM` 声明为 `EEPROM` 类的静态实例。如果你想自己实例化 `EEPROM` 对象进行使用，请如下操作：

```
EEPROMClass eeprom;
```

EEPROM 老版本库只有两个函数，用于访问 AVR MCU 内置的 EEPROM。

`read()`

从 EEPROM 指定的地址处读取 1 B 值。未初始化的位置含有 `255` 值。该函数返回从指定地址读取的字节值。地址从 `0` 开始。

```
uint8_t eeprom.read(int address);
```

`write()`

将 1 B 值按照指定的地址写入微控制器的 EEPROM 存储器。地址从 0 开始。执行一次 EEPROM 写操作大约需要 3.3 ms，AVR EEPROM 能够承受 100 000 次擦写操作，比较耐用。该函数不会返回任何值。

```
void eeprom.write(int address, uint8_t value);
```

最新版本的 EEPROM 库新增了 4 个函数：`update()`、`get()`、`put()`、`EEPROM[]`。EEPROM 库类的源代码值得一读，下面了解代码如何处理任意数据类型。

`update()`

`update()` 函数将 1 B 数据按照指定的地址写入 EEPROM，前提是指定地址处的当前值与传入函数的参数值不同。该函数不会返回任何值。

```
void eeprom.update(int address, uint8_t value);
```

`put()`

`put()` 函数用于将任意类型的数据写入 EEPROM，从指定的地址开始写数据。所写数据可以是基本数据类型（比如 int、float），也可以是结构体。该函数返回数据对象（由 `data` 参数传入）的引用。请注意，这个函数调用 `update()` 函数执行写操作，所以如果指定地址处的数据与传入 `put()` 函数的数据相同，就不会发生写操作。

```
data_ref eeprom.put(int address, data);
```

`get()`

`get()` 函数用于从 EEPROM 中读取并返回任意类型的数据或对象。从 EEPROM 读取的数据被以字节校验码的方式写入 `data argument` 地址处，与 `data` 所指的对象拥有相同大小。该函数返回数据对象（由 data 参数传入）的引用。

```
data_ref eeprom.get(int address, data);
```

`EEPROM[]`

`EEPROM[]` 运算符允许用户像对待数组对象那样访问 EEPROM。真正的 EEPROM 地址是 `EEPROM_base_address + int index`。运算符返回对 EEPROM“单元格”（cell）的引用。

```
data_ref EEPROM[int index]
```

以下简单示例基本上来自 Arduino.cc 官网的 EEPROM 库文档，但带有些许改变。它既向 EEPROM 内存空间写数据，也从中读数据，并且使用求余运算符判断值的奇偶性。

```
#include <EEPROM.h>
// 直接实例化EEPROMClass对象
// 并非在EEPROM.h中做静态声明
EEPROMClass eeprom;
int a = 0;
int value;
void setup()
{
    Serial.begin(9600);
    // 用数据模式预加载EEPROM
    for (int i = 0; i < 512; i++) {
        if (i % 2) // 判断奇偶性
            eeprom.write(i, 0); // 值为奇数
        else
            eeprom.write(i, 1); // 值为偶数
    }
}
// ATmega168拥有512 B大小的EEPROM,ATmega328的EEPROM为1024 B
// 这里只使用512,所以无论哪个Arduino的AVR上都应该是安全的。
```

```
void loop()
{
    value = eeprom.read(a);
    Serial.print(a);
    Serial.print("\t");
    Serial.print(value);
    Serial.println();
    // 变量a在loop()之外声明,所以从调用main()开始,它就一直存在
    a++;
    if (a == 512)
        a = 0;
    delay(500);
}
```

7.1.2 Ethernet

Ethernet 库提供了与 Arduino Ethernet 扩展板进行交互的必需功能。就 Arduino 库来说，Ethernet 库相当复杂，同时提供服务器端与客户端功能。它也支持 4 个并行输入或输出连接，或者它们之一的组合。Ethernet 扩展板使用 SPI 接口与 Arduino 开发板进行通信。

Ethernet 库由 5 个 C++ 类组成，其中大部分类从父类继承，但你不必为大多数应用程序的细节烦恼。不过，如果真的需要查看底层类的定义，那么可以在 Arduino 源代码的 hardware/arduino/avr/cores/arduino 目录下找到。下面列出 Ethernet 库的 5 个类以及每个类的公共成员函数，接下来依次进行介绍。

- `Ethernet` 类
  - `begin()`
  - `localIP()`
  - `maintain()`
- `IPAddress` 类
- `Server` 类
  - `EthernetServer()`
  - `begin()`
  - `available()`
  - `write()`
  - `print()`
  - `println()`
- `Client` 类
  - `EthernetClient()`
  - `connected()`
  - `connect()`
  - `write()`
  - `print()`
  - `println()`
  - `available()`

- `read()`
- `flush()`
- `stop()`

- `EthernetUDP` 类
  - `begin()`
  - `read()`
  - `write()`
  - `beginPacket()`
  - `endPacket()`
  - `parsePacket()`
  - `available()`
  - `stop()`
  - `remotePort()`

1. Ethernet类：EthernetClass

`Ethernet` 类用于初始化 Ethernet 库与网络设置。真正的类名是 `EthernetClass`，在库文件 Ethernet.cpp 中，一个名为 `Ethernet` 的对象被实例化，并在 Ethernet.h 中导出。如果愿意，你可以自己创建 `EthernetClass` 类的实例。当然，也可以直接使用库中提供的实例。

`begin()`

`begin()` 用于初始化 Ethernet 库与创建网络参数。`begin()` 方法（或函数，随你喜欢怎么叫）被重载过，你可以使用 5 种方法调用它。

所有参数都是 unit8_t 字节的数组。对于该方法的 DHCP 形式（`Ethernet.begin(mac)`），若 DHCP 租约成功获取，则返回 `1`，否则返回 `0`。除此之外，该方法的其他形式均不返回任何值。稍后讲解 `IPAddress` 类。

```
int Ethernet.begin(uint8_t *mac);
void Ethernet.begin(uint8_t *mac, IPAddress ip);
void Ethernet.begin(uint8_t *mac, IPAddress ip, IPAddress dns);
void Ethernet.begin(uint8_t *mac, IPAddress ip, IPAddress dns,
                    IPAddress gateway);
void Ethernet.begin(uint8_t *mac, IPAddress ip, IPAddress dns,IPAddress gateway,
                    IPAddress subnet);
```

`localIP()`

该方法用于获取本地机（比如 Ethernet 扩展板）的 IP 地址。使用 DHCP 时，用于确定本地 IP 地址非常有用。如果本地 Ethernet 成功初始化，`Ethernet.localIP()` 将返回一个包含分配或指定的 IP 地址的 `IPAddress` 对象。

```
IPAddress Ethernet.localIP();
```

`maintain()`

老版本的 Ethernet 库并不包含该方法。当一个设备由 DHCP 服务器分配一个 IP 地址时，这被称为“租约”（lease），DHCP 租约有一定的时间限制（具体时间长短取决于 DHCP 服务器是如何配置的）。`Ethernet.maintain()` 方法对 DHCP 租约进行续租。

如果什么都没有发生，`Ethernet.maintain()` 返回 `0`；如果续租失败，则返回 `1`；续租成功，则返回 `2`；若 DHCP 重绑定失败，则返回 `3`；如果重绑定成功，则返回 `4`。

```
int Ethernet.maintain();
```

2. IPAddress类

`IPAddress` 类定义了一个数据对象，其中包含的数据对本地或远程 IP 进行寻址。它拥有 4 种重载的构造函数，每一个接收不同格式的 IP 地址，如下所示：

```
IPAddress()
IPAddress(uint8_t first_octet,
          uint8_t second_octet,
          uint8_t third_octet,
          uint8_t fourth_octet)
IPAddress(uint32_t address)
IPAddress(const uint8_t *address)
```

一个 `IPAddress` 对象能够存放 4 个 IP 地址位组（比如 192.168.1.100，无句点），IP 地址的 32 位整型版本，或者无符号字节数组。`IPAddress` 类用于创建地址数据类型的实例。例如：

```
IPAddress ip(192, 168, 0, 2);
IPAddress dnServer(192, 168, 0, 1);
IPAddress gateway(192, 168, 0, 1);
IPAddress subnet(255, 255, 255, 0);
```

`ip`、`dnServer`、`gateway` 与 `subnet` 是 `IPAddress` 类型的对象。初始化一个 Ethernet 接口时，Ethernet 库知道查找这些名称。请注意，它们初始化都使用多个 8 位组形式。

在 Arduino 源代码的 Arduino/hardware/arduino/avr/cores/arduino 目录下，可以找到 `IPAddress` 的源文件。

3. Server类：EthernetServer

在 Ethernet 术语中，服务器是一个系统（或主机），它接收来自另一个系统的连接请求，并建立通信通道。其中，请求连接的系统称为客户端。服务器被动地等待来自客户端的连接请求，它不会主动发起连接。一个真实的例子是 Web 服务器。Web 服务器等待来自浏览器端的连接请求与页面请求。它会把请求的数据返回到客户端，然后等待下一个请求。当用户在浏览器中点击一个链接、按钮，或者把文本输入到文本框时，就会产生一个请求，并且发送到 Web 服务器。

在 Ethernet 库中，`Server` 类是 `EthernetServer` 的基类。它不被直接调用，而供其他类使用。与 `IPAddress` 一样，`Server` 类也位于 Arduino 源代码的 Arduino/hardware/arduino/avr/cores/arduino 目录之下。

`EthernetServer()`

监听到来自客户端的连接请求时，`EthernetServer()` 方法用于创建要使用的端口。一般来说，实例化一个 `EthernetServer` 对象时，端口即被指定。该方法不会返回任何值（void）。

```
EthernetServer server(int port);
```

示例：

```
EthernetServer newserv = EthernetServer(port);
```

参数 port 可以是 0~65 535 的任何一个数字，但 0~1024 的值一般会被保留下来，用作系统服务，比如 FTP、SSH、Web 服务器。建议使用较大的端口号（比如大于 9000 的端口号），以防出现冲突。

`begin()`

创建好服务器对象之后，调用该方法，使服务器在指定的端口监听来自客户端的连接请求。

```
void newserv.begin();
```

`available()`

返回到客户端的连接，并为通信做好准备。

```
EthernetClient newclient = newserv.available();
```

`write()`

该方法将数据写到连接到服务器的客户端。数据将被写到所有连接的客户端。它接受单个字符（或字节）值，或者指向字符数组的指针，并返回所写字节数。

```
int newserv.write(char data);
int newserv.write(char *data, int size);
```

`print()`

以 ASCII 字符序列形式，打印数据到所连接的服务器。

`println()`

类似于 print()，但输出后会添加一个换行符。该方法可接受的数据类型有字符、字节（uint8_t）、int、long 或 string，并返回打印的字节数。调用时，若不提供参数，则该函数只打印一个简单的换行符。

```
int newserv.println();
int newserv.println(char *data);
int newserv.println(char *data, int BASE);
```

4. Client类：EthernetClient

Client 类创建客户端对象，以便连接到服务器收发数据。服务器与客户端之间进行的数据处理与交换一般由客户端发起。服务器监听与等待客户端的连接，一旦连接建立，客户端就能从服务器请求数据，并发送数据到服务器；或者请求服务器执行一些动作，以满足客户端请求。

在 Ethernet 库中，Client 类是 EthernetClient 类的基类，它不被直接调用，而由 EthernetClient 类继承。与 Server 一样，Client 基类的源代码也位于 Arduino 源代码的 Arduino/hardware/arduino/avr/cores/arduino 目录之下。

`EthernetClient()`

该方法创建客户端对象，以指定的 IP 地址与端口号连接到服务器。connect() 方法定义服务器以及要使用的端口。比如：

```
byte servaddr[] = {172, 120, 40, 10};
EthernetClient newclient;
newclient.connect(servaddr, 80);
```

`connected()`

判断客户端是否连接，其返回值为 `true` 或 `false`。

```
bool newclient.connected();
if (newclient.connected()) {
    //做些什么
}
```

`connect()`

以指定的 IP 地址（4 B 数组）与端口连接到服务器。也可以使用 URL（网址）代替 IP 地址。

```
int newclient.connect();
int newclient.connect(byte *servIP, int port);
int newclient.connect(char *URL, int port);
```

该方法返回一个整型值，表示连接状态。

- `SUCCESS = 1`
- `TIMED_OUT = -1`
- `INVALID_SERVER = -2`
- `TRUNCATED = -3`
- `INVALID_RESPONSE = -4`

`write()`

发送一个值或缓冲区的内容到所连接的服务器，数据以一系列字节的形式发送。`write()` 方法返回所写的字节数。完全可以忽略返回值。

```
int newclient.write(uint8_t value);
int newclient.write(uint8_t *buffer, int len);
```

`print()`

以 ASCII 字符序列形式，打印数据到所连接的服务器。该方法可接受的数据类型有字符、字节（uint8_t）、int、long 或 string。也能指定基数，合法的基数类型有 `BIN`（二进制）、`DEC`（以 10 为基数）、`OCT`（以 8 为基数）、`HEX`（以 16 为基数）。返回打印的字节数。

```
int newclient.print(data);
int newclient.print(data, base);
```

`println()`

类似于 `print()` 方法，但输出 ASCII 字符后会添加一个换行符。调用时，若不提供参数，则该函数会向所连接的服务器发送单个换行符。

```
int newclient.println();
int newclient.println(data);
int newclient.println(data, base);
```

available()

返回从所连接的服务器读取的可用字符数。该方法可以检测是否有输入数据。

```
int newclient.available();
```

read()

从服务器读取下一可用字节。配合使用循环读取多个字符，或者逐个读取并创建每个字符。

```
char newclient.read();
```

flush()

丢弃从服务器接收并存储在缓冲区中的未读字符。

```
void newclient.flush();
```

stop()

从当前连接的服务器断开。一旦断开连接，客户端可以连接到其他服务器（当然也可以再次连接同一个服务器）。

```
void newclient.stop();
```

5. EthernetUDP类

不同于 TCP/IP 这类流协议（没有明确的开始与结束边界），UDP 是数据报协议。数据的每一项都是一个单独的包，叫作数据报，并且数据必须在数据报包的边界之内。UDP 不做错误检测，也不会对数据的传递做保证，只针对非关键数据的短数据包，或者高层软件帮助处理错误检测、重试等操作。UDP 在两台主机之间提供一种快速且相对简单的数据移动方式。

begin()

初始化 UDP 类，在指定端口启动监听，检查是否有输入数据。

```
byte UDP.begin(int port);
```

read()

从指定的缓冲区读取输入数据。若不提供参数，该方法将从当前缓冲区返回 1 个字符。如果指定了缓冲区和大小，read() 方法将从缓冲区返回最多 maxsize 字节。

```
char UDP.read();
char *UDP.read(char *pkt_buffer, int maxsize)
```

请注意，该函数要在调用 UDP.par sePacket() 之后立即使用。

write()

发送数据到远程连接。write() 函数必须在 beginPacket() 与 endPacket() 之间进行调用。beginPacket() 用于初始化数据包，调用 endPacket() 才真正发送数据到远程主机。

```
byte UDP.write(char *message);
byte UDP.write(char *buffer, int size)
```

`beginPacket()`

在指定的 IP 地址与端口，打开连接到远程主机的 UDP 连接。如果连接成功，则返回 `1`（`true`），失败则返回 `0`。

```
int UDP.beginPacket(byte *ip, int port);
```

`endPacket()`

将 `write()` 函数创建的 UDP 数据包发送到远程主机，该主机由 `beginPacket()` 指定。

```
int UDP.endPacket();
```

`parsePacket()`

检查一个打开的 UDP 连接，查看是否有数据报包，并且返回等待数据的大小。请注意，必须在使用 `read()` 或 `available()` 函数获取数据（若有）之前调用 `parsePacket()`。

```
int UDP.parsePacket();
```

`available()`

返回接收缓冲区中当前接收数据的字节数。请注意，只能在调用 `parsePacket()` 之后才可调用 `available()`。

```
int UDP.available();
```

`stop()`

从远程 UDP 主机断开连接，并且释放 UDP 会话占用的所有资源。

```
void USP.stop();
```

`remoteIP()`

返回远程 UDP 连接的 IP 地址，是 4 B 数组。请注意，只有调用 `parsePacket()` 之后才能调用 `remoteIP()` 函数。

```
byte *UDP.remoteIP();
```

`remotePort()`

返回远程 UDP 连接的 UDP 端口，是一个整数值。请注意，只有调用 `parsePacket()` 之后才能调用该函数。

```
int UDP.remotePort();
```

7.1.3 Firmata

Firmata 库非常有意思，拥有巨大的应用潜力。Firmata 提供了使用类似 MIDI 的协议，使得在 Arduino 与主机应用之间进行串行通信成为可能。其开发目标在于，让开发者可以通过主机计算机尽可能多地控制 Arduino 的各个功能。换言之，让使用 Arduino 就像使用主机自身的 I/O 功能扩展一样。

首次启动 Firmata 项目之前，可能想试用演示软件。从旧的 Firmata wiki（http://firmata.org/wiki/Main_Page）可以下载到合适的客户端，Arduino 部分代码已经包含于随 Arduino IDE 一起发布的库中。

本部分只简单讲解 Firmata 库中可用的函数。下面列表给出了库组件的组织方式。然而遗憾的是，这些组件的说明文档似乎不多，为了使用它们，不得不阅读源代码以进行了解。更多细节与用法示例，请参考 Firmata wiki（http://firmata.org，已闲置，不再进行维护），或者查看 Firmata GitHub 存储库（https://github.com/firmata）。应当特别注意协议定义（https://github.com/firmata/protocol），主机应用使用它与运行在 Arduino 上的 Firmata 应用进行通信。

Arduino IDE 提供的 Firmata 库的代码可能不是最新的。请检查 GitHub 存储库（https://github.com/firmata/arduino）。此处文档讲解的函数在你拥有的版本中可能并不存在。

Firmata 库组织如下：

- 基本方法（base methods）
  - `begin()`
  - `printVersion()`
  - `blinkVersion()`
  - `printFirmwareVersion()`
  - `setFirmwareVersion()`
- 发送信息（Sending messages）
  - `sendAnalog()`
  - `sendDigitalPorts()`
  - `sendDigitalPortPair()`
  - `sendSysex()`
  - `sendString()`
  - `sendString()`
- 接收信息（Receiving messages）
  - `available()`
  - `processInput()`
- 回调函数（Callback functions）
  - `attach()`
  - `detach()`
- 消息类型（Message types）

1. 基本方法

接下来，依次学习上面列出的各个函数。

`begin()`

`begin()` 的基本形式用于初始化 Firmata 库，并把串行数据速率设置为默认的 57 600 波特率。第二种形式接收一个 long 类型的参数，指定 Firmata 与主机系统通信所需的波特率。第三种形式使用流启动库，而非 `Serial`。它可以和任何实现了 Stream 接口（比

如 Ethernet、Wi-Fi 等）的数据流一起工作。关于该方法的更多信息，请访问 Firmata 的 GitHub 页面（https://github.com/firmata/arduino/issues）参与讨论。

```
void Firmata.begin();
void Firmata.begin(long);
void Firmata.begin(Stream &s)
```

`printVersion()`

发送库协议版本到主计算机。

```
void Firmata.printVersion();
```

`blinkVersion()`

引脚 13（Arduino 上的板载 LED）上的协议版本。

```
void Firmata.blinkVersion();
```

`printFirmwareVersion()`

发送固件名与版本到主计算机。

```
void Firmata.printFirmwareVersion();
```

`setFirmwareVersion()`

使用程序文件名（不包含扩展名），设置固件名与版本。

```
void Firmata.setFirmwareVersion(const char *name,
                                byte vers_major, byte vers_minor);
```

2. 发送信息

`sendAnalog()`

发送模拟数据信息。

```
void Firmata.sendAnalog(byte pin, int value);
```

`sendDigitalPort()`

以单字节发送数字端口状态。

```
void Firmata.sendDigitalPort(byte pin, int portData);
```

`sendString()`

发送字符串到主计算机。

```
void Firmata.sendString(const char* string);
```

`sendString()`

使用自定义命令类型发送字符串到主计算机。

```
void Firmata.sendString(byte command, const char* string);
```

`sendSysex()`

发送包含任意字节数组的命令。

```
void Firmata.sendSysex(byte command, byte bytec, byte* bytev);
```

3. 接收信息

`available()`

在输入缓冲区中查看是否存在任何输入消息。

```
int Firmata.available();
```

`processInput()`

从输入缓冲区获取并处理输入消息，然后发送数据到已注册的回调函数。

```
void Firmata.processInput();
```

4. 回调函数

为了将函数附加到特定的消息类型，该函数必须与回调函数相匹配。Firmata 中有 3 种基本的回调函数：generic、string、sysex，接下来依次学习。还有一种回调函数用于处理系统重置。回调函数如下：

`attach()`

将函数附加到一个特定的输入信息类型。

```
void attach(byte command, callbackFunction newFunction);
void attach(byte command, systemResetCallbackFunction newFunction);
void attach(byte command, stringCallbackFunction newFunction);
void attach(byte command, sysexCallbackFunction newFunction);
```

`detach()`

将函数从特定的输入信息类型分离。

```
void Firmata.detach(byte command);
```

5. 消息类型

一个函数可以被附加到特定的消息类型。Firmata 支持如下消息类型。

`ANALOG_MESSAGE`

单个引脚的模拟值

`DIGITAL_MESSAGE`

8 位数字引脚数据（一个端口）

`REPORT_ANALOG`

启用 / 关闭模拟引脚报告

`REPORT_DIGITAL`

启用 / 关闭数字引脚报告

`SET_PIN_MODE`

在 INPUT/OUTPUT/PWM/ 之间修改引脚模式。

`FIRMATA_STRING`

对于 C 类型字符串，使用 `stringCallbackFunction` 函数类型。

`SYSEX_START`

对于一般情况，不定长信息（借助 MIDI SysEx 协议）；使用 `sysex CallbackFunction` 函数类型。

`SYSTEM_RESET`

将固件重置为默认状态，使用 `systemResetCallbackFunction` 函数类型。

7.1.4 GSM

GSM 库和 GSM 扩展板一起用于连接到 GSM/GPRS 网络。Arduino IDE 1.0.4 及之后的版本均包含该库。GSM 库支持大部分函数，用于操作 GSM 手机中的各种功能，比如接打电话、收发 SMS，以及通过 GPRS 网络连接到互联网。GSM 表示全球移动通信系统，GRPS 是 General Packet Radio Service（通用分组无线服务）的缩写。

GSM 扩展板集成了调制解调器，将数据从串口传送到 GSM 网络。调制解调器利用 AT 命令执行各种功能。通常，每个 AT 命令都是更长系列命令的一部分，用于执行某项特定功能。GSM 库依赖 SoftwareSerial 库，在 Arduino 与 GSM 调制解调器之间提供通信支持。

GSM 库是最近才加入的，如果你正使用较旧版本的 IDE，那么可能没有这个库。在 IDE 中查看可用的库，了解你是否拥有可用的 GSM 库。

GSM 库类套装比较复杂，其功能的完整介绍超出本书范围。接下来的部分中，我们只对其功能做些简单的介绍。更多相关细节，请阅读 Arduino GSM 库的有关说明，或者查看 Arduino IDE 中内置的的帮助文档。一些厂商——比如 Adafruit——也生产与 Arduino 兼容的 GSM 扩展板，他们为自己的产品提供相应的库。

1. Ethernet库的兼容性

GSM 库很大程度上与目前的 Arduino Ethernet 库相兼容，比如将使用 Arduino Ethernet 或 Wi-Fi 库的程序移植为 GSM 库，并与 GSM 扩展板一起使用，这相对比较容易。当然，仍需要对针对特定库的代码做一些较小的修改，比如包含 GSM 与 GPRS 特定的库，以及从蜂窝网络提供商获取网络设置。

2. 库组织结构

GSM 库相当复杂，由 10 个基本类组成。下面列出了每个 GSM 类中的函数。

- `GSM` 类
  - `begin()`
  - `shutdown()`
- `GSMVoiceCall` 类
  - `getVoiceCallStatus()`
  - `ready()`
  - `voiceCall()`
  - `answerCall()`
  - `hangCall()`
  - `retrieveCallingNumber()`

- GSM_SMS 类
  - beginSMS()
  - ready()
  - endSMS()
  - available()
  - remoteNumber()
  - read()
  - write()
  - print()
  - peek()
  - flush()
- GPRS 类
  - attachGPRS()
- GSMClient 类
  - ready()
  - connect()
  - beginWrite()
  - write()
  - endWrite()
  - connected()
  - read()
  - available()
  - peek()
  - flush()
  - stop()
- GSMServer 类
  - ready()
  - beginWrite()
  - write()
  - endWrite()
  - read()
  - available()
  - stop()
- GSMModem 类
  - begin()
  - getIMEI()
- GSMScanner 类
  - begin()
  - getCurrentCarrier()
  - getSignalStrength()
  - readNetworks()

- GSMPIN 类
  - begin()
  - isPIN()
  - checkPIN()
  - checkPUK()
  - changePIN()
  - switchPIN()
  - checkReg()
  - getPINUsed()
  - setPINUsed()
- GSMBand 类
  - begin()
  - getBand()
  - setBand()

3. GSM类

该类提供与调制解调器进行通信的函数。它管理扩展板的连接，对 GSM 基础设施执行必要的系统注册。所有 Arduino GSM/GPRS 程序需要包含该类的一个对象，用于处理低层通信功能。

基于 GSM 的所有功能都以该类为基类，它应该如下这样进行实例化：

```
GSM gsmbase;
```

`begin()`

启动 GSM/GPRS 调制解器，并附加到 GSM 网络。`begin()` 方法的完整原型如下：

```
begin(char* pin=0, bool restart=true, bool synchronous=true);
```

`begin()` 方法拥有 4 种不同的调用方式，每一个参数都被指派一个默认值。第一种形式不带参数，假设 SIM 没有配置引脚。

```
gsmbase.begin();
gsmbase.begin(char *pin);
gsmbase.begin(char *pin, bool restart);
gsmbase.begin(char *pin, bool restart, bool sync);
```

`shutdown()`

关闭调制解调器（断电）。

```
gsmbase.shutdown();
```

4. GSMVoiceCall类

`GSMVoiceCall` 类通过调制解调器启动语音通信，由麦克风、扬声器以及连接到 GSM 扩展板的少量电路提供。

所有用于接收语音、创建语音通话的 GSM 功能都以该类为基类，其实例化方法如下：

```
GSMVoiceCall gsmvc;
```

`getVoiceCallStatus()`

返回语音通话状态，共有 IDLE_CALL、CALLING、RECEIVINGCALL、TALKING 这 4 种状态。

```
GSM3_voiceCall_st getVoiceCallStatus();
gsmvc.getVoiceCallStatus();
```

`ready()`

返回最后命令的执行状态。若成功执行，则返回 1；若正在执行，则返回 0；若发生错误，则返回大于 1 的值。

```
int ready();
gsmvc.ready();
```

`voiceCall()`

以同步或异步模式发起语音通话。如果是异步模式，那么电话响时，voiceCall() 即返回。在同步模式下，当通话建立或取消时，voiceCall() 才返回。

第一个参数是一个字符串，其中含有待呼叫的号码，地方分机号可用可不用。在 voiceCall() 完成之前（命令完成），缓冲区不应该被释放或占用。参数 timeout 的单位为 ms，仅用在同步模式中。如果设置为 0，则 voiceCall() 将会一直等待，直到另一端接起电话。

```
int voiceCall(const char* to, unsigned long timeout=30000);
gsmvc.voiceCall(to);           // 使用默认超时
gsmvc.voiceCall(to, timeout);  // 指定超时
```

`answerCall()`

接受传入的语音通话。在异步模式下，如果最后命令仍在执行，answerCall() 返回 0；如果成功执行，则返回 1；若发生错误，则返回大于 1 的值。在同步模式下，如果呼叫有应答，则 answerCall() 返回 1，否则返回 0。

```
gsmvc.answerCall();
```

`hangCall()`

挂断建立的通话或来电响铃。在异步模式下，如果最后命令仍在执行，hangCall() 返回 0；如果成功执行，则返回 1；若发生错误，则返回大于 1 的值。在同步模式下，如果呼叫有应答，则 hangCall() 返回 1，否则返回 0。

```
gsmvc.hangCall();
```

`retrieveCallingNumber()`

获取主叫号码，并将其放入缓冲区。参数 buffer 是一个指向字符缓冲区的指针，bufsize 指定缓冲区的大小，单位为字节。缓冲区应该足够大，至少能够容纳 10 个字符。

在异步模式下，如果最后命令仍在执行，retrieveCallingNumber() 返回 0；如果成功执行，则返回 1；若发生错误，则返回大于 1 的值。在同步模式下，如果正确获得号码，则 retrieveCallingNumber() 返回 1，否则返回 0。

```
int retrieveCallingNumber(char* buffer, int bufsize);
gsmvc.retrieveCallingNumber(buffer, bufsize);
```

5. GSM_SMS类

该类提供收发 SMS 的功能。

`beginSMS()`

定义要接收 SMS 的电话号码。电话号码是一个字符数组。在异步模式下，如果最后命令仍在执行，则返回 `0`；如果成功执行，则返回 `1`；若发生错误，则返回大于 `1` 的值。在同步模式下，如果前面的命令成功执行，则返回 `1`，否则返回 `0`。

```
int SMS.beginSMS(char *phone_number)
```

`ready()`

返回最后 GSM SMS 命令的状态。在异步模式下，如果最后命令仍在执行，则 `ready()` 返回 `0`；如果成功执行，则返回 `1`；若发生错误，则返回大于 `1` 的值。在同步模式下，如果前面的命令成功执行，则返回 `1`，否则返回 `0`。

```
int SMS.ready()
```

`endSMS()`

通知调制解调器信息已经编写好，并做好发送准备。在异步模式下，如果最后命令仍在执行，则返回 `0`；如果成功执行，则返回 `1`；若发生错误，则返回大于 `1` 的值。在同步模式下，如果成功执行，则返回 `1`，否则返回 `0`。

```
int SMS.endSMS()
```

`available()`

如果有 SMS 可读，则该函数返回信息中的字符数，否则返回 `0`。

```
int SMS.available()
```

`remoteNumber()`

从传入的 SMS 提取远程电话号码，并以字符数组形式返回。size 参数定义传递给 `remoteNumber()` 函数的数组的最大大小。在异步模式下，如果仍在执行，则该函数返回 `0`；如果成功执行，则返回 `1`；若发生错误，则返回大于 `1` 的值。在同步模式下，如果成功执行，则该函数返回 `1`，否则返回 `0`。

```
int SMS.remoteNumber(char *remote_phone, int number_size)
```

`read()`

从 SMS 读 1 B（一个字符）。以整型形式返回读取的字节，如果无数据可读，则返回 `-1`。

```
int SMS.read()
```

`write()`

向 SMS 写入 1 B 大小的字符。

```
int write(int character)
```

`print()`

将字符数组写到 SMS，并返回成功写入的字节数。

```
int SMS.print(char *message)
```

`peek()`

从 SMS 返回下一可用字节（一个字符），不会移除字符，若无可用数据则返回 `-1`。

```
int SMS.peek()
```

`flush()`

发送完所有输出字符后，清空调制解器中的所有发送的信息。

```
void SMS.flush()
```

6. GPRS类

所有 GPRS 功能都建立在 GPRS 类之上，包括网络客户端与服务器功能。该类也负责包含 TCP 通信涉及的文件。

`attachGPRS()`

连接到给定的接入点（APN），发起 GPRS 通信。移动运营商有 APN，就像在蜂窝网络与因特网之间架起一座“桥梁”。该函数返回如下字符串之一：`ERROR`、`IDLE`、`CONNECTING`、`GSM_READY`、`GPRS_READY`、`TRANSPARENT_CONNECTED`。

```
char *GPRS.attachGPRS(char *apn, char *user, char *password)
```

7. GSMClient类

该类用于创建客户端，它可以连接到服务器并收发数据。

`ready()`

返回最后命令的状态。在异步模式下，若最后命令仍在执行，则返回 `0`；若已成功执行，则返回 `1`；若发生错误，则返回大于 `1` 的值。在同步模式下，若命令成功执行，则返回 `1`，否则返回 `0`。

```
int GSMClient.ready()
```

`connect(char *IP, int port)`

连接到指定 IP 地址的指定端口。如果连接成功，则返回 `true`，否则返回 `false`。

```
bool GSMClient.connect(char *hostip, int port)
```

`beginWrite()`

向所连接的服务器启动一个写操作。

```
void GSMClient.beginWrite()
```

`write()`

向所连接的服务器写数据。返回所写数据的字节数。

```
int GSMClient.write(char data)
int GSMClient.write(char *data)
int GSMClient.write(char *data, int size)
```

`endWrite()`

停止向所连接的服务器写数据。

```
void GSMClient.endWrite()
```

connected()

返回一个客户端的连接状态。请注意，如果连接已经被关闭，但缓冲区中仍有未读数据，那么客户端仍然被认为处于连接状态。如果有客户端连接，则返回 true，否则返回 false。

```
bool GSMClient.connected()
```

read()

从客户端连接的服务器读取下一字节可用数据。返回下一字节，若无可用数据，则返回 -1。

```
int GSMClient.read()
```

available()

从连接的服务器返回待读的字节数。

```
int GSMClient.available()
```

peek()

从输入信息中返回下一字节数据，不会从输入缓冲区中移除。连续调用 peek() 将简单返回相同的字节。

```
int GSMClient.peek()
```

flush()

丢弃当前在输入缓冲区中等待的任何数据，并将可用数据计数重置为 0。

```
void GSMClient.flush()
```

stop()

强制从服务器断开连接。

```
void GSMClient.stop()
```

8. GSMServer类

GSMServer 类用于创建服务器，该服务器可以发送数据到连接的客户端，也可以从所连接的客户端接收数据。它实现了网络服务器的功能，与 Ethernet 与 Wi-Fi 库类似。请注意，一些网络运营商不允许从他们的网络外部传入网络连接。

ready()

返回最后命令的状态。在异步模式下，若最后命令仍在执行，该函数返回 0；若已成功执行，则返回 1；若发生错误，则返回大于 1 的值。在同步模式下，若命令执行成功，则返回 1，否则返回 0。

```
int GSMServer.ready()
```

beginWrite()

向所有连接的客户端开始写操作。

```
void GSMServer.beginWrite()
```

`write()`

将数据写到所连接的客户端。返回所写的字节数。

```
int GSMServer.write(char data)
int GSMServer.write(char *data)
int GSMServer.write(char *data, int size)
```

`endWrite()`

停止向连接的客户端写数据。

```
void GSMServer.endWrite()
```

`read()`

从连接的客户端读下一字节可用数据。返回读取的字节，若无可用数据，则返回 `-1`。

```
int GSMServer.read()
```

`available()`

监听来自客户端的连接请求。返回连接的客户端数量。

```
int GSMServer.available()
```

`stop()`

停止服务器监听客户端连接请求。

```
void GSMServer.stop()
```

9. GSMModem类

`GSMModem` 类为内部 GSM 调制解调器提供诊断支持函数。

`begin()`

检查调制解调器的状态并重启。该函数必须在 `getIMEI()` 之前被调用。若调制解调器工作正常，则返回 `1`，否则会产生错误。

```
int GSMModen.begin()
```

`getIMEI()`

查询调制解调器，以获取其 IMEI（国际移动设备识别码）编号，并以字符串形式返回。该函数只能在 `begin()` 之后才能调用。

```
char *GSMModen.getIMEI()
```

10. GSMScanner类

`GSMScanner` 类提供一系列函数，用户获取诊有关网络与运营商的诊断信息。

`begin()`

重置调制解调器。若调制解调器运行正常，则返回 `1`，否则返回错误代码。

```
int GSMSScanner.begin()
```

`getCurrentCarrier()`

以字符串形式返回当前网络服务器提供商的名称。

```
char *GSMSScanner.getCurrentCarrier()
```

`getSignalStrength()`

以字符串形式返回网络连接的相对信号强度，用 ASCII 数字 0~31 表示（31 表示功率大于 51 dBm）。如果未检测到信号，则返回 99。

```
char *GSMSScanner.getSignalStrength()
```

`readNetworks()`

搜索可用的网络运营商。该函数返回一个字符串，包含检测到的运营商列表。

```
char *GSMSScanner.readNetworks()
```

11. GSMPIN类

GSMPIN 类包含一系列实用工具函数，用于与 SIM 卡进行通信。

`begin()`

重置调制解调器。如果调制解调器运行正常，则返回 1，否则返回错误代码。

```
int GSMPIN.begin()
```

`isPIN()`

检测 SIM 卡，判断它是否被 PIN 锁定。如果 PIN 锁处于关闭状态，则返回 0；若处于开启状态，则返回 1。若 PUK 锁处于开启状态，则返回 -1；若遇到错误，则返回 -2。

```
int GSMPIN.isPIN()
```

`checkPIN()`

查询带有 PIN 的 SIM 卡，判断是否合法。如果 PIN 合法，则返回 0，否则返回 -1。

```
int GSMPIN.checkPIN(char *PIN)
```

`checkPUK()`

查询 SIM，判断 PUK 码是否合法，并创建新的 PIN 妈。若成功，则返回 0，否则返回 -1。

```
int GSMPIN.checkPUK(char *PUK, char *PIN)
```

`changePIN()`

验证旧 PIN 合法后，更改 SIM 卡的 PIN 码。

```
void GSMPIN.changePIN(char *oldPIN, char *newPIN)
```

`switchPIN()`

更改 PIN 锁状态。

```
void GSMPIN.switchPIN(char *PIN)
```

`checkReg()`

检测并判断调制解调器是否注册到 GSM/GPRS 网络。若已注册，则返回 0；若漫游，则返回 1；若遇到错误，则返回 -1。

```
int GSMPIN.checkReg()
```

`getPINUsed()`

检测并判断是否使用 PIN 锁。若是，则返回 `true`，否则返回 `false`。

```
bool GSMPIN.getPINUsed()
```

`setPINUsed()`

设置 PIN 锁状态。如果参数是 `true`，则启用 PIN 锁，否则不启用。

```
void GSMPIN.setPINUsed(bool used)
```

12. GSMBand类

`GSMBand` 类提供关于调制解器连接的频带信息，其中也包含设置频带的方法。

`begin()`

重置调制解调器。如果调制解调器操作成功，则返回 `1`，否则返回错误代码。

```
int GSMBand.begin()
```

`getBand()`

返回当前调制解器连接所用的频带。

```
char *GSMBand.getBand()
```

`setBand()`

设置调制解调器所用的频带。

```
bool GSMBand.setBand(char *band)
```

7.1.5 LiquidCrystal

该类允许 Arduino 开发板控制液晶显示器（LCD）模块。它专门针对基于 Hitachi HD44780（或者相兼容的）芯片组的 LCD，大多数基于文本的 LCD 都使用 Hitachi HD44780 芯片组。LiquidCrystal 库支持 4 位或 8 位接口模式，使用 Arduino 的 3 个引脚连接 RS（寄存器选择）、时钟使能（clock enable）、R/W（读 / 写）控制线。

`LiquidCrystal()`

该方法用于创建 `LiquidCrystal` 类的实例对象，它有几种不同的形式，分别针对不同的 LCD 接口方法。

```
LiquidCrystal(uint8_t rs, uint8_t enable, uint8_t d0, uint8_t d1,
              uint8_t d2, uint8_t d3, uint8_t d4, uint8_t d5,
              uint8_t d6, uint8_t d7);

LiquidCrystal(uint8_t rs, uint8_t rw, uint8_t enable, uint8_t d0,
              uint8_t d1, uint8_t d2, uint8_t d3, uint8_t d4,
              uint8_t d5, uint8_t d6, uint8_t d7);

LiquidCrystal(uint8_t rs, uint8_t rw, uint8_t enable, uint8_t d0,
              uint8_t d1, uint8_t d2, uint8_t d3);

LiquidCrystal(uint8_t rs, uint8_t enable, uint8_t d0, uint8_t d1,
              uint8_t d2, uint8_t d3);
```

各个参数含义如下：

rs：连接到 LCD RS 引脚的 Arduino 引脚
rw：连接到 LCD RW 引脚的 Arduino 引脚
enable：连接到 LCD 使能引脚的 Arduino 引脚
d0~d7：连接到 LCD 数据引脚的 Arduino 引脚

其中，d4、d5、d6、d7 信号线是可选的。如果只使用 4 根数字线，则可以不使用 d4、d5、d6、d7。

例如：

```
LiquidCrystal lcd(12, 11, 10, 5, 4, 3, 2);
```

`begin()`

初始化连接 LCD 控制器（位于 LCD 模块上）的接口。参数用于指定 LCD 的宽度与高度。默认字符大小为 5 × 8 像素。该函数必须在使用其他任何 LCD 库函数之前被调用。

```
void lcd.begin(uint8_t cols, uint8_t rows,
               uint8_t charsize = LCD_5x8DOTS)
```

`clear()`

清空 LCD 屏幕，并将光标重置到屏幕左上角。

```
void lcd.clear()
```

`home()`

将光标定位到 LCD 屏幕的左上角。使用该函数并不会清空 LCD 屏幕，若想清空 LCD 屏幕，请使用 `clear()` 函数。

```
void home()
```

`setCursor()`

将光标定位到参数 `column` 与 `row` 指定的位置。

```
void setCursor(uint8_t column, uint8_t row)
```

`write()`

向 LCD 写 1 B（或字符）数据，返回所写字节数。

```
size_t write(uint8_t)
```

`print()`

其实，该函数是标准 Arduino 运行时 AVR-GCC 代码的一部分，并且被重载过，以便接收不同类型的数据。Print.h 文件位于 hardware/arduino/avr/cores/arduino/Print.h 目录之下，其中定义了不同形式的 `print()` 函数，如下所示：

```
size_t print(const __FlashStringHelper *);
size_t print(const String &);
size_t print(const char[]);
size_t print(char);
size_t print(unsigned char, int = DEC);
```

```
size_t print(int, int = DEC);
size_t print(unsigned int, int = DEC);
size_t print(long, int = DEC);
size_t print(unsigned long, int = DEC);
size_t print(double, int = 2);
size_t print(const Printable&);
```

`cursor()`

将 LCD 屏幕上的光标（下划线字符）显示在下一个字符要写的位置。

```
void cursor()
```

`noCursor()`

禁用光标，其实是隐藏。不会对下一个字符要显示的位置产生影响。

```
void noCursor()
```

`blink()`

使光标闪动。

```
void blink()
```

`noBlink()`

关闭光标闪动。

```
void noBlink()
```

`display()`

若最初 LCD 被 `noDisplay()` 函数禁用，则可使用 `display()` 函数启用。这样即可恢复之前可见的光标与文本，或者恢复从禁用屏幕以来添加、删除、修改的内容。

```
void display()
```

`noDisplay()`

禁用 LCD 显示，且不改变屏幕上任何现有文本。

```
void noDisplay()
```

`scrollDisplayLeft()`

在屏幕上向左滚动文本一个字符的距离。

```
void scrollDisplayLeft()
```

`scrollDisplayRight()`

在屏幕上向右滚动文本一个字符的距离。

```
void scrollDisplayRight()
```

`autoscroll()`

启用自动滚动。向屏幕添加文本时，该函数会把原有字符向左或向右移动一个字符的距离，具体移动方向取决于当前文本的书写方向。

```
void autoscroll()
```

`noAutoscroll()`

禁用 LCD 的自动滚动函数。

```
void noAutoscroll()
```

`leftToRight()`

启用自动滚动功能时，设置文本移动的方向为从左到右。

```
void leftToRight()
```

`rightToLeft()`

启用自动滚动功能时，设置文本移动的方向为从右到左。

```
void rightToLeft()
```

`createChar()`

创建 5 × 8 像素的字符。该字符使用字节数组定义，每行 1 个。每字节只使用 5 个最低有效位。

```
void createChar(uint8_t, uint8_t[])
```

7.1.6 SD

SD 库对读写 SD 闪存卡提供支持，包含全尺寸与微型 SD 类型（就接口与功能而言，它们完全相同，只是大小不同而已）。SD 库基于 William Greiman 编写的 sdfatlib 库创建而来。

该库将 SD 卡看作一个带有 FAT16 或 FAT32 文件系统的小磁盘。它使用短文件名（8.3 格式）。传递给 SD 库函数的文件名可能包含路径，目录名由斜杠隔开（像 Linux，不使用 MS-DOS 与 Windows 中使用的反斜杠）。

SPI 接口与 SD 卡进行通信。在标准 Arduino 开发板上，使用数字引脚 11、12、13。一个额外引脚用作选择引脚，一般是 10 号引脚，当然也可以指定另一个引脚承担该功能。请注意，即便选用了另外一个引脚作为选择引脚，SS 引脚（10 号引脚）也必须保留，用作库工作时的输出。

1. SD类

SD 类提供了一系列函数，用于访问 SD 卡，以及操纵其中的文件与目录。

`begin()`

初始化 SD 库与 SD 卡接口。`csPin` 为可选参数，定义用作选择引脚的引脚，默认使用 10 号引脚（`SD_CHIP_SELECT_PIN`）。该函数必须在使用其他任何 SD 函数之前调用。若执行成功，则返回 `true`，否则返回 `false`。

```
bool SD.begin(uint8_t csPin = SD_CHIP_SELECT_PIN);
```

`exists()`

检测指定的文件或目录在 SD 卡上是否存在。字符串 `filepath` 可以是完全限定路径名（FQPN）。若指定文件或目录存在，则返回 `true`，否则返回 `false`。

```
bool SD.exists(char *filepath);
```

`mkdir()`

在 SD 卡上创建目录。它也会创建任何必需的中间目录。若目录创建成功，则返回 `true`，否则返回 `false`。

```
bool SD.mkdir(char *filepath);
```

`open()`

打开 SD 卡上的文件进行读写。若不提供 `mode` 参数，则默认以读模式打开文件。`open()` 函数返回一个 `File` 对象，该对象可以被作为 Boolean 值进行测试。若指定文件不能被打开，则 `File` 求值结果为 `false`。有 `FILE_READ` 与 `FILE_WRITE` 两种模式可用。

```
File SD.open(const char *filename, uint8_t mode = FILE_READ);
```

`remove()`

从 SD 卡删除（移除）一个文件。`filepath` 为 FQPN。如果移除成功，则返回 `true`，否则返回 `false`。

```
bool SD.remove(char *filepath);
```

`rmdir()`

从 SD 卡移除空目录。若目录成功删除，则返回 `true`；若发生错误（比如目录非空），则返回 `false`。

```
bool SD.rmdir(char *filepath);
```

2. File类

`File` 类提供一系列函数，用于读写 SD 卡中的单独文件。`File` 对象由 `SD.open()` 函数创建，如下所示：

```
fname = SD.open("data.txt", FILE_WRITE);
```

File 对象提供了许多方法用于操纵文件内容。

`available()`

返回文件中的可读字节数。

```
int fname.available()
```

`close()`

关闭一个文件，并确保所有剩余数据都事先写入文件。

```
void fname.close()
```

`flush()`

将文件缓冲区中的所有剩余数据全部写入文件。不会关闭文件。

```
void fname.flush()
```

`peek()`

从一个文件读取 1 B 数据，且不会移动内部数据指针。连续调用 `peek()` 函数将返回相同字节。

```
int fname.peek()
```

position()

返回文件内的当前位置，下一字节将从该位置开始读或写。

```
uint32_t fname.position()
```

print()

将数据写到一个文件，该文件处于“写打开”模式。print() 可以接受的数据类型有字符型（char）、字节型（uint8_t）、整型（int）、长整型（long）、字符串类型（string）。使用 print() 时，可以自己指定基数，合法基数类型有 BIN（二进制）、DEC（十进制）、OCT（八进制）、HEX（十六进制）。返回所写的字节数。

```
int fname.print(data)
int fname.print(char *data, int BASE)
```

注意：示例中显示的是字符串数据。

println()

将数据写到一个处于“写打开”模式的文件，并在最后添加回车换行符。println() 可以接受的数据类型有字符型（char）、字节型（uint8_t）、整型（int）、长整型（long）、字符串类型（string）。使用 print() 时，可以自己指定基数，合法基数类型有 BIN（二进制）、DEC（十进制）、OCT（八进制）、HEX（十六进制）。返回所写的字节数。

```
int fname.println()
int fname.println(data)
int fname.println(data, int BASE)
```

seek()

移动文件中的内部指针到新位置。新位置必须在 0 到文件大小之间，包含文件大小。若执行成功，则返回 true；若有错误发生（比如查找的位置超出文件末尾），则返回 false。

```
bool fname.seek(uint32_t pos)
```

size()

以字节形式返回文件大小。

```
uint32_t fname.size()
```

read()

从文件读取下一字节数据。若无数据可读，则返回 -1。

```
int fname.read(void *buf, uint16_t nbyte)
```

write()

将数据写到文件。该函数接受的数据可以是单字节，也可以是数据对象（1 B、一个字符或字符串）。size 参数指定要写入 SD 卡的数据量，它返回所写的字节数。

```
size_t fname.write(uint8_t)
size_t fname.write(const uint8_t *buf, size_t size)
```

isDirectory()

若 fname 对象是一个目录，则返回 true，否则返回 false。

```
bool fname.isDirectory()
```

`openNextFile()`

打开目录中的下一个文件或文件夹，并返回一个 `File` 对象的新实例。

```
File openNextFile(uint8_t mode = O_RDONLY)
```

`rewindDirectory()`

与 `openNextFile()` 一起使用，该函数返回一个目录中的第一个文件或子目录。

```
void fname.rewindDirectory()
```

7.1.7 Servo

Servo 库提供一系列函数，用于控制伺服电机，比如遥控飞机中的那些电机。创建 `Servo` 类的实例之后，接着可以使用 `attach()` 函数传入引脚号，将伺服电机连接到指定引脚。控制伺服电机的脉冲在后台产生。该类实例化方法如下：

```
Servo servo;
```

`attach()`

将伺服电机附加到 I/O 引脚。第二种形式允许调用者指定最小与最大脉冲宽度，单位为 ms。若执行成功，则返回通道号（channel number），否则返回 `0`。

```
uint8_t servo.attach(int pin)
uint8_t servo.attach(int pin, int min, int max)
```

`write()`

设置伺服电机角度，单位为度。若 value 值大于 `200`，就会被视作单位为 ms 的脉冲宽度。

```
void servo.write(int value)
```

`read()`

返回最后写入的脉冲宽度，角度取值范围为 0°~180°。

```
int servo.read()
```

`writeMicroseconds()`

设置伺服电机脉冲宽度，单位为 ms。

```
void servo.writeMicroseconds(int value)
```

`readMicroseconds()`

返回伺服电机的当前脉冲宽度，单位为 ms。

```
int servo.readMicroseconds()
```

`attached()`

若 servo 对象已经被附加到物理伺服电机上，则返回 `true`。

```
bool servo.attached()
```

`detach()`

停止一个已经附加的 servo 对象在为其指派的 I/O 引脚上产生脉冲。

```
void servo.detach()
```

7.1.8 SPI

SPI 库支持使用串行外围接口（SPI）总线，以便与兼容 SPI 的外围设备进行通信。最具有代表性的是带有内置 SPI 接口的芯片，它也可以用于在两个微控制器之间进行通信。

`SPISettings` 类配置 SPI 端口。参数被合并到一个单独的 `SPISettings` 对象，并传递给 `SPI.Begin Transaction()`。

```
SPISettings(uint32_t clock, uint8_t bitOrder, uint8_t dataMode)
```

示例：

```
SPISettings spiset(uint32_t clock,
                   uint8_t bitOrder,
                   uint8_t dataMode)
```

`beginTransaction()`

使用 `SPISettings` 对象中定义的设置初始化 SPI 接口。

```
void SPI.beginTransaction(SPISettings)
```

`endTransaction()`

停止与 SPI 接口通信。通常在选择引脚失效并允许其他库使用 SPI 接口之后调用。

```
void SPI.endTransaction()
```

`usingInterrupt()`

SPI 通信发生在中断上下文中时，调用该函数注册中断号。

```
void SPI.usingInterrupt(uint8_t interruptNumber)
```

`begin()`

启动 SPI 库并初始化 SPI 接口。设置 SCK、MOSI、SS 引脚为输出模式，设置 SS 为高电平时，将 SCK 与 MOSI 拉为低电平。

```
void SPI.begin()
```

`end()`

禁用 SPI 接口，但保留引脚模式（in 或 out）不变。

```
void SPI.end()
```

`transfer()`

通过 SPI 接口传送（接收或发送）1 B 数据。

```
uint8_t SPI.transfer(uint8_t data)
```

`setBitOrder()`

设置转移到 SPI 接口的位序，有两种位序可选，分别为 `LSBFIRST`（低位优先）与 `MSBFIRST`（高位优先）。该函数不应该在新项目中使用。使用 `beginTransaction()` 函数配置 SPI 接口。

```
void SPI.setBitOrder(uint8_t bitOrder)
```

`setClockDivider()`

设置 SPI 时钟分频器，相对于系统时钟。对于基于 AVR 的 Arduino 开发板，有效因子为 2、4、8、16、32、64、128。该函数不应该在新项目中使用。使用 `beginTransaction()` 函数配置 SPI 接口。

```
void SPI.setClockDivider(uint8_t clockDiv)
```

`setDataMode()`

设置 SPI 接口的时钟极性与相位。该函数不应该在新项目中使用。使用 `beginTransaction()` 函数配置 SPI 接口。

```
void SPI.setDataMode(uint8_t dataMode)
```

7.1.9 SoftwareSerial

SoftwareSerial 库在 Arduino 的数字 I/O 引脚上实现了基于软件的串行通信。换言之，它采用"位拆裂"（bit-banger）技术，模拟了传统串口。当需要使用一个以上的串口，而 AVR 微控制器内置的 USART 被指派执行其他一些功能（比如 USB 接口）时，它非常有用。

`SoftwareSerial` 库支持多串口，每个串口的速度最高可达 115 200 位 /s。使用 `SoftwareSerial` 的多个实例时，一次只能有一个接收数据。使用的 I/O 引脚必须支持引脚更改中断。`SoftwareSerial` 为每个串口实例提供 64 B 的接收缓冲区。

创建 `SoftwareSerial` 对象之后，即可将其与其他串行 I/O 操作一起使用。类构造函数传递的数字引脚分别用于输入（rx）与输出（tx）。

```
SoftwareSerial(uint8_t rxPin, uint8_t txPin, bool inv_logic = false)
```

示例：

```
SoftwareSerial serial = SoftwareSerial(rxPin, txPin)
```

`available()`

返回串口缓冲区中可读的字节数。

```
int serial.available()
```

`begin()`

设置串口波特率（速度），有效波特率有 300、600、1200、2400、4800、9600、14 400、19 200、28 800、31 520、57 600、115 200。

```
void serial.begin(long speed)
```

`isListening()`

测试串口，判断其是否正在监听输入。若接口主动等待输入，则返回 `true`，否则返回 `false`。

```
bool serial.isListening()
```

`overflow()`

如果输入超出 `SoftwareSerial` 对象中 64 B 大小的接收缓冲区，就会设置一个标志。调用 `overflow()` 函数将返回该标记。若返回值为 `true`，则表明发生溢出。调用 `overflow()` 函数清除标记。

```
bool serial.overflow()
```

`peek()`

从串行输入缓冲区返回最旧字符，但不会将其移除。后续调用将返回相同字符。若缓冲区中无字节，则 `peek()` 返回 `-1`。

```
int serial.peek()
```

`read()`

从接收缓冲区返回一个字符，并将其移除。`read()` 函数通常用在循环中。若无数据可读，则返回 `-1`。请注意，任意给定时刻只能有一个 `SoftwareSerial` 实例可以接收输入数据。`listen()` 函数选择要激活的接口。

```
int serial.read()
```

`print()`

`print()` 函数行为表现类似于 `Serial.print()` 函数，它能接收 `Serial.print()` 可以接收的任何数据类型，包括字符（char）、字节（byte）、整型（int）、长整型（long）、字符串型（string），返回所写的字节数。

```
int serial.print(data)
```

`println()`

与 `serial.print()` 函数功能完全相同，只是最后会向输出添加回车换行符（CR/LF）。返回所写字节数。若无数据可写，则将简单地打印一个 CR/LF。

```
int serial.println(data)
```

`listen()`

为 `SoftwareSerial` 实例开启监听（数据接收）。任意给定时刻只能有一个 `SoftwareSerial` 实例对象可以接收数据，并且调用实例对象的 `listen()` 函数，成为活动监听器。来自其他接口实例的数据会被丢弃。

```
bool serial.listen()
```

`write()`

从串口传送数据（原始字节）。其行为类似于 `Serial.write()` 函数，返回所写字节数。

```
size_t serial.write(uint8_t byte)
```

7.1.10 Stepper

Stepper 库用于控制单极和双极步进电机，它们都带有合适硬件以处理所需电流。

Stepper 库拥有两种形式的构造函数，一种针对单极电机，另一种针对双极电机。每一个都创建 `Stepper` 类的一个新实例。它在程序开始时，即在 `setup()` 与 `loop()` 函数之前被调用。`steps` 参数定义了电机输出轴旋转一圈的步数，`pin1`、`pin2`、`pin3`、`pin4` 参数指定要使用的数字引脚。

```
Stepper(int steps, int pin1, int pin2);
Stepper(int steps, int pin1, int pin2, int pin3, int pin4);
```

示例：

```
Stepper stepdrv = Stepper(100, 3, 4);
```

`setSpeed()`

设置速度（步进率），即所谓的最大功率转速（RPM）。这不会引起电机转动，只是设置调用 `step()` 函数时要使用的速度。

```
void stepdrv.setSpeed(long speed);
```

`step()`

命令电机移动指定的步数。正步数让电机向一个方向转，负步数使其向相反方向转。

```
void stepdrv.step(int stepcount);
```

7.1.11 TFT

TFT（薄膜晶体管）库提供了一系列函数，用于在 TFT 屏幕上绘制文本、图像、图形。Arduino 1.0.5 以及之后版本包含该库。TFT 库简化了在 TFT 屏幕上显示图形的过程。它基于 Adafruit ST7735H 库，你在 GitHub 上可以找到它（http://bit.ly/ada-st7735）。ST7735H 库又基于 Adafruit GFX 库，同样可以在 GitHub 上找到（http://bit.ly/ada-gfx）。

TFT 库被设计为，与使用 AVR 微控制器 SPI 通信功能的接口一起使用。如果 TFT 扩展板含有 SD 卡插槽，那么通过来自 Arduino 的单独的选择信号，SD 库可以读写数据。TFT 库依赖于 SPI 库，实现与屏幕以及 SD 卡的通信。并且，所有使用 TFT 库的程序都需要包含 SPI 库。

1. TFT类

`TFT` 类拥有两种形式的构造函数，一种在使用标准 Arduino SPI 引脚（SPI 硬件）时使用，另一种允许自由指定要使用的引脚。

```
TFT(uint8_t cs, uint8_t dc, uint8_t rst)
TFT(uint8_t cs, uint8_t dc, uint8_t mosi, uint8_t sclk, uint8_t rst)
```

参数含义如下：

cs：芯片选择引脚

dc：数据或命令模式选择

rst：重置引脚
mosi：若不使用 SPI 硬件，则用作 MOSI 的引脚
sclk：若不使用 SPI 硬件，则用作时钟的引脚

示例：

```
#define cs  10
#define dc  9
#define rst 8
TFT disp = TFT(cs, ds, rst);
```

TFT 库的 Esplora 版本使用预定义引脚。使用起来非常简单，所有需要做的只是实例化一个 TFT 对象。

```
EsploraTFT disp = EsploraTFT;
```

`begin()`

调用该函数将会对 TFT 库组件进行初始化。该函数必须在其他任何函数之前调用。`begin()` 通常在程序的 `setup()` 函数中调用。

```
void disp.begin()
```

`background()`

使用纯色覆写整个显示屏幕。该函数也可以用于清空屏幕。请注意，其实屏幕无法显示每种颜色的 256 个级别，而是使用 5 位表示蓝色与红色，6 位表示绿色。

```
void disp.background(uint8_t red, uint8_t green, uint8_t blue)
```

`stroke()`

向屏幕绘制之前调用，用于设置线条与边框的颜色。类似于 `background()` 函数，`stroke()` 使用 5 位表示蓝色与红色，6 位表示绿色。

```
void disp.stroke(uint8_t red, uint8_t green, uint8_t blue)
```

`noStroke()`

移除所有轮廓描边颜色。

```
void disp.noStroke()
```

`fill()`

设置屏幕上对象与文本的填充色。类似于 `stroke()` 函数，`fill()` 使用 5 位表示蓝色与红色，6 位表示绿色。

```
void disp.fill(uint8_t red, uint8_t green, uint8_t blue)
```

`noFill()`

禁用对象与文本颜色填充。

```
void disp.noFill()
```

`setTextSize()`

设置文本大小，通过调用 `text()` 函数写出该文本。默认文本大小为 1 或 10 像素。每次

增加文本大小都会导致屏幕上文本高度增加 10 像素。

```
void disp.setTextSize(uint8_t size)
```

`text()`

将文本写到屏幕指定的位置。调用 `text()` 之前，可以调用 `fill()` 函数设置文本颜色。

```
void disp.text(const char * text, int16_t x, int16_t y);
```

`point()`

在屏幕指定位置绘制一个点。点的颜色通过提前调用 `fill()` 函数进行设置。

```
void disp.point(int16_t x, int16_t y)
```

`line()`

在起点与终点之间绘制一条直线，线条颜色由 `stroke()` 函数进行设置。

```
void disp.line(int16_t x1, int16_t y1, int16_t x2, int16_t y2)
```

`rect()`

以给定的点（矩形左上顶点）、宽度、高度绘制一个矩形。

```
void disp.rect(int16_t x, int16_t y, int16_t width, int16_t height)
```

`width()`

返回 TFT 屏幕宽度，单位为像素。

```
int disp.width()
```

`height()`

返回 TFT 屏幕高度，单位为像素。

```
int disp.height()
```

`circle()`

以给定的圆心（x，y）与半径（r）在屏幕上绘制圆形。

```
int disp.circle(int16_t x, int16_t y, int16_t r)
```

`image()`

将从 SD 卡加载的图像绘制到屏幕指定位置。

```
void image(PImage image, int xpos, int yPos)
```

`loadImage()`

使用提供的图像文件名创建一个 `PImage` 实例对象。请注意，图像文件必须是 24 位 BMP 类型，并且位于 SD 卡根目录下。使用 `PImage` 函数 `loadImage()`。

```
PImage disp.loadImage(char *imgname)
```

2. PImage

`PImage` 类包含的一系列函数用于封装图像，以及在 TFT 屏幕绘制位图图像。

`PImage.height()`

将一幅图像封装在一个 `PImage` 对象中后，可能需要查询以获取其高度。该函数整型值形式返回图像对象的高度。

`PImage.width()`

以整型值形式返回封装的图像对象的宽度。

`PImage.isValid()`

若图像对象包含合法的位图文件，则返回 `true`，否则返回 `false`。

7.1.12 Wi-Fi

Wi-Fi 库让 Arduino 拥有连接无线网络的能力。此处不详细讲解 Wi-Fi 库中所有可用的函数，其中许多函数与 Ethernet 库中的函数相似或者完全一样。Arduino IDE 早期版本的内置帮助中没有 Wi-Fi 库参考页面（遗憾是，写作本书之时，一些 Linux 发行包中所带的 Arduino IDE 都是早期版本），而后来版本中有。Wi-Fi 库的源代码看上去的确随老版本的 IDE 一起安装了，至少在我的 Kubuntu 开发系统中是这样的。

Wi-Fi 库与 SPI 库一起使用，用以与 Wi-Fi 模块以及可选的 SD 内存卡进行通信。基线型（baseline-type）Arduino（见第 4 章）使用 SPI 引脚 10、11、12、13 与 Wi-Fi 扩展板进行通信，而 Mega 类型开发板则使用引脚 10、50、51、52。此外，在 Arduino Wi-Fi 扩展板上，引脚 7 被用作 Arduino 与 Wi-Fi 扩展板的握手信号，因此不能将其留作他用。其他 Wi-Fi 扩展板可能也有类似的约束。

Arduino Wi-Fi 扩展板可以充当一个服务器，用于接收入站连接（incoming connection）；或者充当客户端，创建与现有服务器的连接。Wi-Fi 库提供 WEP 与 WPA2 个人加密模式，但不支持 WPA2 企业加密。此外，如果服务器节点不广播其 SSID（服务集标识符），那么 Wi-Fi 扩展板就不能创建连接。

与 Ethernet 库类似，Wi-Fi 库由 5 个 C++ 类组成。其中大多数类从父类继承，大可不必为大多数应用的细节而烦恼。不过，如果需要了解 `Client`、`Server`、`UDP` 以及其他类的底层定义，可以在 Arduino 源代码的 libraries/WiFi 目录中找到它们。下面列出了 Wi-Fi 库的 5 个类，以及每个类的公共成员函数。

- `Wi-Fi` 类
  - `begin()`
  - `disconnect()`
  - `config()`
  - `setDNS()`
  - `SSID()`
  - `BSSID()`
  - `RSSI()`
  - `encryptionType()`
  - `scanNetworks()`
  - `getSocket()`
  - `macAddress()`

- IPAddress 类
  - localIP()
  - subnetMask()
  - gatewayIP()
- Server 类
  - WiFiServer()
  - begin()
  - available()
  - write()
  - print()
  - println()
- Client 类
  - WiFiClient()
  - connected()
  - connect()
  - write()
  - print()
  - println()
  - available()
  - read()
  - flush()
  - stop()
- UDP 类
  - begin()
  - available()
  - beginPacket()
  - endPacket()
  - write()
  - parsePacket()
  - peek()
  - read()
  - flush()
  - stop()
  - remoteIP()
  - remotePort()

Arduino Wi-Fi 扩展板基于 HDG204 802.11b/g 芯片。请注意，其他 Wi-Fi 扩展板（比如 Adafruit Wi-Fi 扩展板基于 TI CC3000 WiFi 芯片）可能会为特定的 Wi-Fi 芯片使用不同的库。其中许多功能与这里列出的功能类似，但仍然有一些不同之处需要考量。在 GitHub（http://bit.ly/ada-cc3000）上可以下载 Adafruit 库。请访问 Adafruit 网站（http://bit.ly/ada-cc3000-wifi）以获取更多细节。

7.1.13 Wi-Fi类

下面简单介绍一下 `Wi-Fi` 类。有关函数功能的讲解，请参考 Ethernet 库。

`Wi-Fi` 类包含的函数用于初始化库与网络设置。在 Wi-Fi.h 包含文件中，可以找到该类的定义。

```
WiFiClass()

int begin(char* ssid)
int begin(char* ssid, uint8_t key_idx, const char* key)
int begin(char* ssid, const char *passphrase)

int disconnect()

void config(IPAddress local_ip)
void config(IPAddress local_ip, IPAddress dns_server)
void config(IPAddress local_ip, IPAddress dns_server, IPAddress gateway)
void config(IPAddress local_ip, IPAddress dns_server,
            IPAddress gateway, IPAddress subnet)

void setDNS(IPAddress dns_server1)
void setDNS(IPAddress dns_server1, IPAddress dns_server2)

char* SSID()
char* SSID(uint8_t networkItem)
uint8_t* BSSID(uint8_t* bssid)
int32_t RSSI()
int32_t RSSI(uint8_t networkItem)

uint8_t encryptionType()
uint8_t encryptionType(uint8_t networkItem)
int8_t scanNetworks()

static uint8_t getSocket()
uint8_t* macAddress(uint8_t* mac)
static char* firmwareVersion()
uint8_t status()
int hostByName(const char* aHostname, IPAddress& aResult)
```

7.1.14 IPAddress类

与 `Ethernet` 库中的 `IPAddress` 类类似，Wi-Fi 库中的 `IPAddress` 类为网络配置有关的信息提供了一个容器。

```
IPAddress localIP()
IPAddress subnetMask()
IPAddress gatewayIP()
```

7.1.15 Server类

`Server` 类用于创建服务器，接收来自客户端的连接请求，进而交换数据。客户端可能是另一个带有 Wi-Fi 扩展板的 Arduino、一台桌面 PC、笔记本电脑，或者是任何一台带有 Wi-Fi 功能的设备。请阅读 7.1.2 节中有关 `print()` 与 `println()` 的讲解。

```
WiFiServer(uint16_t)
WiFiClient available(uint8_t* status = NULL)

void begin()
int print(data)
int print(data, base)

int println()
int println(data)
int println(data, base)

size_t write(uint8_t)
size_t write(const uint8_t *buf, size_t size)

uint8_t status()
```

7.1.16 Client类

`Client` 类用于创建 Wi-Fi 客户端，以连接到服务器，进而收发数据。此处所说的“服务器”可以是另一个带有 Wi-Fi 扩展板的 Arduino、桌面 PC、笔记本电脑，或者其他任何一种具备 Wi-Fi 服务器功能的设备。请阅读 7.1.2 节中有关 `print()` 与 `println()` 的讲解。

```
WiFiClient()
WiFiClient(uint8_t sock)

uint8_t connected()

int connect(IPAddress ip, uint16_t port)
int connect(const char *host, uint16_t port)

size_t write(uint8_t)
size_t write(const uint8_t *buf, size_t size)

int print(data)
int print(data, base)

int println()
int println(data)
int println(data, base)

int available()
int read()
int read(uint8_t *buf, size_t size)
int peek()

void flush()
void stop()
```

7.1.17 UDP类

`UDP` 类使用 UDP 协议收发短信息。与 TCP/IP 这类流协议（即没有明确的开始与结束边界）不同，UDP 是数据报协议。数据的每一项都是一个单独的包，叫作数据报，并且数据必须在数据报包的边界之内。UDP 不做错误检测，也不会对数据的传递做保证，只针对非关键

数据的短数据包，或者高层软件帮助处理错误检测、重试等操作。UDP 在两台主机之间提供一种快速且相对简单的数据移动方式。

```
WiFiUDP()

uint8_t begin(uint16_t)
void stop()

int beginPacket(IPAddress ip, uint16_t port)
int beginPacket(const char *host, uint16_t port)
int endPacket()

size_t write(uint8_t)
size_t write(const uint8_t *buffer, size_t size)

int parsePacket()
int available()
int read()
int read(unsigned char* buffer, size_t len)
int peek()
void flush()

IPAddress remoteIP()
uint16_t remotePort()
```

7.1.18 Wire

Wire 库与 TWI- 或 I2C 类型的设备进行通信。更多有关 AVR 微控制器的 TWI 功能的信息，请参考第 2 章与第 3 章。第 8 章将讲解一些使用 I2C 与 Arduino 进行通信的扩展板。

表 7-1 列出了不同类型 Arduino 开发板上 TWI 引脚的位置。有关内容请参考第 4 章中开发板引脚的布局图。

表7-1：不同类型Arduino开发板上TWI引脚的位置

开发板	SDA	SCL
Uno、Ethernet	A4	A5
Mega2560	20	21
Leonardo	2	3

Wire 库的核心类是 `TwoWire` 类。

示例：

```
TwoWire twi = TwoWire()
```

`begin()`

初始化 TWI 库，并以主机模式或从机模式激活 I2C 接口。若未指定地址，则 I2C 接口默认采用主机模式。

```
void twi.begin()
void twi.begin(uint8_t addr)
void twi.begin(int addr)
```

`requestFrom()`

主接口使用该函数向从设备请求数据。使用 `available()` 与 `read()` 函数获取数据字节。该函数返回从指定设备读取的数据字节数。

```
uint8_t twi.requestFrom(uint8_t addr, uint8_t quantity)
uint8_t twi.requestFrom(uint8_t addr, uint8_t quantity, uint8_t stop)
uint8_t twi.requestFrom(int addr, int quantity)
uint8_t twi.requestFrom(int addr, int quantity, int stop)
```

`beginTransmission()`

向指定地址处的 I2C 从设备开始传送数据。先使用 `write()` 函数将数据放入传输队列，然后使用 `endTransmission()` 函数进行传送。

```
void twi.beginTransmission(uint8_t addr)
void twi.beginTransmission(int addr)
```

`endTransmission()`

先将由 `write()` 函数存入队列的数据传送到从设备，然后结束数据传送（由 `beginTransmission()` 函数发起）。

```
uint8_t twi.endTransmission()
uint8_t twi.endTransmission(uint8_t stop)
```

`write()`

将提供的数据写入传输队列，以便从主设备发送到从设备；或者从从设备发送到主设备，以响应数据请求。该函数返回写入队列的字节数。

```
size_t twi.write(uint8_t data)
size_t twi.write(const uint8_t *data)
size_t twi.write(const uint8_t *data, size_t len)
```

`available()`

返回可供 `read()` 函数读取的字节数。调用 `requestFrom()` 函数之后，由主设备进行调用，或者发生数据接收事件之后由从设备调用。

```
int twi.available()
```

`read()`

从由主设备发送到从设备（或者从从设备发送到主设备）的数据中读取 1 B。

```
int twi.read()
```

`onReceive()`

注册从设备从主设备接收数据时要调用的函数（处理器函数）。

```
void twi.onReceive(void (*)(int))
```

`onRequest()`

注册主设备向从设备请求数据时要调用的函数。

```
void twi.onRequest(void (*)(void))
```

7.1.19 Esplora

Arduino Esplora 库提供了一系列函数，借助 `Esplora` 类，用户可以轻松连接 Esplora 开发板与各种传感器、驱动器。有关引脚信息请参考第 4 章。

Esplora 开发板可用的传感器如下所示：

- 双轴模拟摇杆
- 摇杆中心按钮
- 四按钮组
- 麦克风
- 光传感器
- 温度传感器
- 三轴加速度传感器
- 双 TinkerKit 输入连接器

Esplora 开发板上可用的驱动器有：

- 明亮 RGB（红 – 绿 – 蓝）LED
- 压电式蜂鸣器
- 双 TinkerKit 输出连接器

`Esplora()`

创建 Esplora 类的实例对象。

```
Esplora esp = Esplora()
```

`readSlider()`

返回整数值，对应于滑块控件的当前位置。取值范围为 0~1023。

```
unsigned int esp.readSlider()
```

`readLightSensor()`

返回整型值，对应于照射到 Esplora 开发板上光线传感器的光线数量。

```
unsigned int esp.readLightSensor()
```

`readTemperature()`

返回一个有符号整数，对应于当前周围环境的温度，以℉或℃表示。`scale` 参数可取 `DEGREES_C` 或 `DEGREES_F`。温度范围为 −40℃ ~150℃或者 −40 ℉ ~302 ℉。

```
int esp.readTemperature(const byte scale);
```

`readMicrophone()`

返回整型值，取值范围为 0~1023，对应于麦克风检测到的周围噪声的大小。

```
unsigned int esp.readMicrophone()
```

`readJoystickSwitch()`

读取摇杆按钮，并且返回 `0` 或 `1023`。它与 `read JoystickButton()` 函数可以相互替换。

```
unsigned int esp.readJoystickSwitch()
```

`readJoystickButton()`

读取摇杆按钮，并且返回 `Low` 或 `HIGH`（按下或非按下）。该函数功能与 `readJoystickSwitch()` 函数相同，但它返回的值与 `readButton()` 函数相同。

```
bool readJoystickButton()
```

`readJoystickX()`

返回摇杆 x 轴的位置，取值范围为 -512~512。

```
int esp.readJoystickX()
```

`readJoystickY()`

返回摇杆 y 轴的位置，取值范围为 -512~512。

```
int esp.readJoystickY()
```

`readAccelerometer()`

返回所选坐标轴的当前值。`axis` 参数可以取的值有 `X_AXIS`、`Y_AXIS`、`Z_AXIS`。若返回值为 `0`，则表示加速度传感器垂直于重力方向；返回的正值或负值表示加速度的方向与加速率。

```
int esp.readAccelerometer(const byte axis)
```

`readButton()`

读取 Esplora 开发板上特定按钮的当前状态。若按钮被按下，则返回一个低值（`false`），否则返回一个高值（`true`）。

```
bool esp.readButton(byte channel)
```

`writeRGB()`

写入一系列值，分别用于定义 Esplora RGB LED 中红色、绿色、蓝色的亮度水平。

```
void esp.writeRGB(byte red, byte green, byte blue)
```

`writeRed()`

接受 0~255 的数值，定义红色 LED 的亮度。

```
void esp.writeRed(byte red)
```

`writeGreen()`

接受 0~255 的数值，定义绿色 LED 的亮度。

```
void esp.writeGreen(byte green)
```

`writeBlue()`

接受 0~255 之间的数值，定义蓝色 LED 的亮度。

```
void esp.writeBlue(byte blue)
```

`readRed()`

返回为红色 LED 最后一次设置的亮度值。

```
byte esp.readRed()
```

`readGreen()`

返回为绿色 LED 最后一次设置的亮度值。

```
byte esp.readGreen()
```

`readBlue()`

返回为蓝色 LED 最后一次设置的亮度值。

```
byte esp.readBlue()
```

`tone()`

以给定的频率从 Esplora 板载蜂鸣器发出一个音调。若不提供 `duration` 参数则会一直发声，直到调用 `noTone()` 函数终止。请注意，一次只能使用一个频率，并且使用 `tone()` 函数将干扰控制红色 LED 的程度。

```
void esp.tone(unsigned int freq)
void esp.tone(unsigned int freq, unsigned long duration)
```

`noTone()`

停止向蜂鸣器输出方波信号。

```
void esp.noTone()
```

1. USB库

核心 USB 库允许用户将 Arduino Leonardo 或 Micro 用作鼠标或键盘设备，以连接到主计算机。

> 如果鼠标或键盘库一直处于运行状态，将会对 Arduino 编程造成困难。`Mouse.move()` 与 `Keyboard.print()` 这类函数应当只在主机准备好处理它们的时候调用。针对此问题，一种解决方法是，Arduino 发送鼠标或键盘信息时，使用控制系统或物理开关进行控制。

2. Mouse

鼠标函数允许 Leonardo 或 Micro 控制主机上的光标运动。返回的光标位置不是绝对的，而总是相对于上一个位置的。

```
Mouse.begin()
Mouse.click()
Mouse.end()
Mouse.move()
Mouse.press()
Mouse.release()
Mouse.isPressed()
```

3. Keyboard

键盘函数允许 Leonardo 或 Micro 发送击键到所连接的主机。然而，并非每一个 ASCII 字符都能被 Keyboard 库发送，比如非打印字符，但 Keyboard 库支持使用修改键（modifier keys）。

```
Keyboard.begin()
Keyboard.end()
Keyboard.press()
Keyboard.print()
Keyboard.println()
Keyboard.release()
Keyboard.releaseAll()
Keyboard.write()
```

同时按下某些键与修改键时，修改键会改变这些键的行为。表 7-2 列出了 Leonardo 支持的修改键。

表7-2：USB键盘修改键

修改键	十六进制值	十进制值	修改键	十六进制值	十进制值
KEY_LEFT_CTRL	0x80	128	KEY_PAGE_UP	0xD3	211
KEY_LEFT_SHIFT	0x81	129	KEY_PAGE_DOWN	0xD6	214
KEY_LEFT_ALT	0x82	130	KEY_HOME	0xD2	210
KEY_LEFT_GUI	0x83	131	KEY_END	0xD5	213
KEY_RIGHT_CTRL	0x84	132	KEY_CAPS_LOCK	0xC1	193
KEY_RIGHT_SHIFT	0x85	133	KEY_F1	0xC2	194
KEY_RIGHT_ALT	0x86	134	KEY_F2	0xC3	195
KEY_RIGHT_GUI	0x87	135	KEY_F3	0xC4	196
KEY_UP_ARROW	0xDA	218	KEY_F4	0xC5	197
KEY_DOWN_ARROW	0xD9	217	KEY_F5	0xC6	198
KEY_LEFT_ARROW	0xD8	216	KEY_F6	0xC7	199
KEY_RIGHT_ARROW	0xD7	215	KEY_F7	0xC8	200
KEY_BACKSPACE	0xB2	178	KEY_F8	0xC9	201
KEY_TAB	0xB3	179	KEY_F9	0xCA	202
KEY_RETURN	0xB0	176	KEY_F10	0xCB	203
KEY_ESC	0xB1	177	KEY_F11	0xCC	204
KEY_INSERT	0xD1	209	KEY_F12	0xCD	205

7.2 第三方库

针对 Arduino 开发板，有许多第三方库可以使用。其中一些由个人创建，另一些由销售 Arduino 硬件与配件的公司创建。此外，一些经销商也提供各种库，为自己的产品提供软件支持。搜索相应网站或者查看 eBay 列表，一般会发现这些库代码。

表 7-3~ 表 7-9 列出了一系列第三方库，并划分为不同类别。对于这些库的介绍一般都很简单，相关资料也不够丰富，我们无法详细讲解，也无法对其做出评判。本书一直力求简洁、紧凑，所以此处不再讲解这些库，更多细节请参考链接 http://www.arduino.cc/en/Reference/Libraries。

表7-3：通信（网络与协议）

库	描述
Messenger	处理来自计算机的基于文本的消息
NewSoftSerial	改进版 SoftwareSerial 库
OneWire	控制采用 One Wire 协议的设备（来自美国达拉斯半导体公司）
PS2Keyboard	从 PS2 键盘读取字符
Simple Message System	在 Arduino 与计算机之间发送消息。
SSerial2Mobile	使用手机发送文本消息或电子邮件（基于 SoftwareSerial 之上的 AT 命令）
Webduino	扩展 Web 服务器库（针对 Arduino Ethernet 扩展板）
X10	在 AC 电源线上发送 X10 信号
XBee	以 API 模式与 XBee 通信
SerialControl	基于串行连接对其他 Arduino 进行远程控制

表7-4：感知

库	描述
Capacitive Sensing	将两个或更多引脚转换为电容式传感器
Debounce	读取噪声数字输入（比如来自按钮）

表7-5：显示与LED

库	描述
GFX	带有标准图形例程的基类（由 Adafruit Industries 开发）
GLCD	为基于 KS0108 或相同芯片的 LCD 提供图形例程
LedControl	控制 LED 矩阵或带有 MAX7221 或 MAX7219 的七段数码管
LedControl	用于取代 Matrix 库，驱动带有 Maxim 芯片的多 LED
LedDisplay	控制 HCMS-29xx，滚动 LED 显示屏
Matrix	基本的 LED 矩阵显示控制库
PCD8544	用于类 Nokia 55100 显示器上的 LCD 控制器（由 Adafruit Industries 开发）
Sprite	基本图像精灵控制库，用在在 LED 矩阵中显示动画
ST7735	用于 1.8″，128 × 160 TFT 屏幕上的 LCD 控制器（由 Adafruit Industries 开发）

表7-6：声音与信号波形

库	描述
FFT	针对声音或其他模拟信号进行频率分析
Tone	在微控制器任意一个引脚上产生背景声音频率方波

表7-7：发动机与PWM

库	描述
TLC5940	控制 TLC5940 IC，16 通道、12 位 PWM

表7-8：计时

库	描述
DateTime	该库记录软件中当前日期与时间
Metro	以特定时间间隔为动作计时
MsTimer2	使用计时器 2 中断每隔 *N* ms 触发一个动作

表7-9：实用工具

库	描述
PString	打印到缓冲区的轻量级类
Streaming	简化打印语句的库

第 8 章

扩展板

Arduino 扩展板是一个扩展电路板，它被设计为与标准 Arduino 开发板（比如 Uno、Duemilanove、Leonardo、Mega）上的连接器一起使用。扩展板上的引脚与 Arduino 相连，所以 DC 电源、数字 I/O、模拟 I/O 等都可以提供给扩展板。本章将介绍一些与 Arduino 开发板兼容的常用扩展板，第 10 章将讲解如何制作自定义扩展板。

扩展板可以应用于各种场景，比如驱动发动机控制器的小型开发板、Ethernet 接口、SD 闪存与显示设备。许多扩展板支持堆叠能力，允许在一个 Arduino 开发板上一次连接两个或更多扩展板。

本章内容会涉及许多销售商与厂商，但请注意，这不是有意为它们做宣传。此处提到的扩展板都很常用，非常有代表性，并且对于任意一种类型的扩展板，你都能找到很多销售商。它们销售的扩展板都是一样的，或者是等效的产品。选购时，请货比三家，自行选择。

本章并没有全部列出各种可用的扩展板。此外，也有一些家庭手工业者，他们专门制作各种现有扩展板的变形，以及一些以前从未出现过的新扩展板。我们介绍扩展板时，所选的都是常用且有广泛代表性的扩展板，同时提供了相关链接。参考这些链接，你可以学到更多相关内容，或者花钱购买一两个扩展板玩玩。有时，我在书中补充内容部分对于某个扩展板提供了很多细节信息，这些信息大都来自扩展板供应商，但这并不意味着我特别喜欢某款扩展板。我只是喜欢文档资料，因为从中能得到自己需要的信息，以便继续做我想做的事。你可能也会处于这样的境地：拥有一个看似非常有用的扩展板，但相关资料很少或干脆没有，又或者相关说明资料都是英文的（或者其他一些你不懂的语言）。希望本章内容能够填补这些空缺，至少让你大致知道去哪里找。

书中列出的一些扩展板已经无法从原供应商那里买到，但你仍然可以利用其他渠道。大多数供应商都提供了可用的文档链接，所以如果发现一个相似的扩展板（毕竟硬件是开源的），你通常都能得到自己所需的技术信息。

当你寻找一个扩展板时，需要记住，有些人好像并不知道扩展板是什么。它不是一侧带有一排引脚的模块（相关内容参见第 9 章）。扩展板是带有许多物理特性的开发板，这些物理特性将在 8.2 节进行讲解。除此之外，其他方面都可以被看作一个模块，有些模块可以直接插到 Arduino 上（通常不是“插上”这么简单，还需要用某种连线将电源与信号发送到 Arduino 开发板适当的引脚上），而有些则不能。

8.1 扩展板的电气特性

比较本章中各种扩展板的引脚图，你会注意到一种相同模式：采用双线接口（TWI 或 I2C 接口）的扩展板总是使用 Arduino 的 A4 与 A5 引脚。在一个基线型（baseline-type）开发板（Diecimila、Duemilanove、Uno R2 或 SMD）上，你将在模拟 I/O 连接器以及更新型的扩展布局板（Uno R3、Ethernet、Leonardo）上看到它们。信号也会出现在扩展引脚头上（在 USB 或 RJ45 连接器的上边角）。采用 I2C 的扩展板能够利用任一组引脚，可以认为，若不采用一些巧妙的编程方法，则 A4 与 A5 不能用作其他用途。

同样，采用 SPI 接口的扩展板通常会使用 D10、D11、D12、D13 引脚（依次为 SS、MOSI、MISO、SCK）。其中，D10 用作选择，是可选的，一些扩展板使用其他数字引脚实现该功能。请记住，每一个从设备通过 SPI 接口从主设备接收输入数据之前，必须使用选择信号明确将其选中。若扩展板上连接的 SPI 设备多于一个，可能需要使用一个以上的数字 I/O 引脚进行信号选择。

采用 UART（Atmel 称其为 USART）的扩展板通常使用引脚 D0 与 D1（依次为 Rx 与 Tx，在 Mega 类型的 Arduino 上分别为 RxD0 与 TxD0）。一些蓝牙扩展板采用 UART 接口与蓝牙模块进行通信，类似于采用 RS-232 与 RS-485 接口的扩展板。

还有一些扩展板几乎会用到 Arudino 的每一个独立引脚。8.4.22 节介绍的 DIY 多功能扩展板就是如此，但它的确有引脚连接到开发板上未被使用的信号（此处是三路数字信号和一路模拟信号）。一般而言，你可以放心假设，I/O 扩展板将使用大多数或全部可用的 Arduino 引脚，而用于支持显示的扩展板一般不会为访问未使用的信号提供任何连接点。基于这个原因，这些类型的扩展板一般都是非堆叠型的，它应该用于 Arduino 一系列堆叠扩展板的最顶层。

大多数扩展板的电路并不十分复杂，反而相对简单，构建在现有 IC 之上或者一些组件之上。类似于 Arduino 开发板，扩展板本质上是各种 IC、继电器等组件的载体。扩展板的电气特性就是组成它的各种芯片或组件的电气特性。简单的设计有助于降低扩展板的开发成本，而其使用的 IC 或组件提供的功能让扩展板非常有用。

最后，一些扩展板可能会使用主动电路或设备（比如光隔离器或继电器）对进出 Arduino 的信号进行缓存，但它们中的大部分只是简单用作安装连接器或组件的地方，比如 8.4.2 节介绍的扩展板，它们就使用多针连接器。这些连接（可能是插口，也可能是排针）只是

Arduino 自身引脚的扩展而已，一般没有保护措施能够阻止用户将 12 V 接到 5 V（或者 3.3 V）数字输入上。很显然，这种错误的行为会烧坏 Arduino 上的 AVR 微控制器。请一定记得检查电压以及 AVR 微控制器对电流的限制。

8.2　扩展板的物理特性

从物理外形看，标准扩展板与基准 Arduino 开发板一样宽（大小见第 4 章），其长度可以与 Arduino 一样，也可以比 Arduino 更长或更短。当然，也可以增加扩展板的宽度，使其比基准 Arduino 开发板更宽。唯一的约束是，扩展板上的引脚必须与底部 Arduino 开发板上的引脚插口对应（引脚布局及大小请参考第 4 章）。安装扩展板非常简单，只需将扩展板上的排针插入 Arduino 开发板对应的插口即可，如图 8-1 所示。

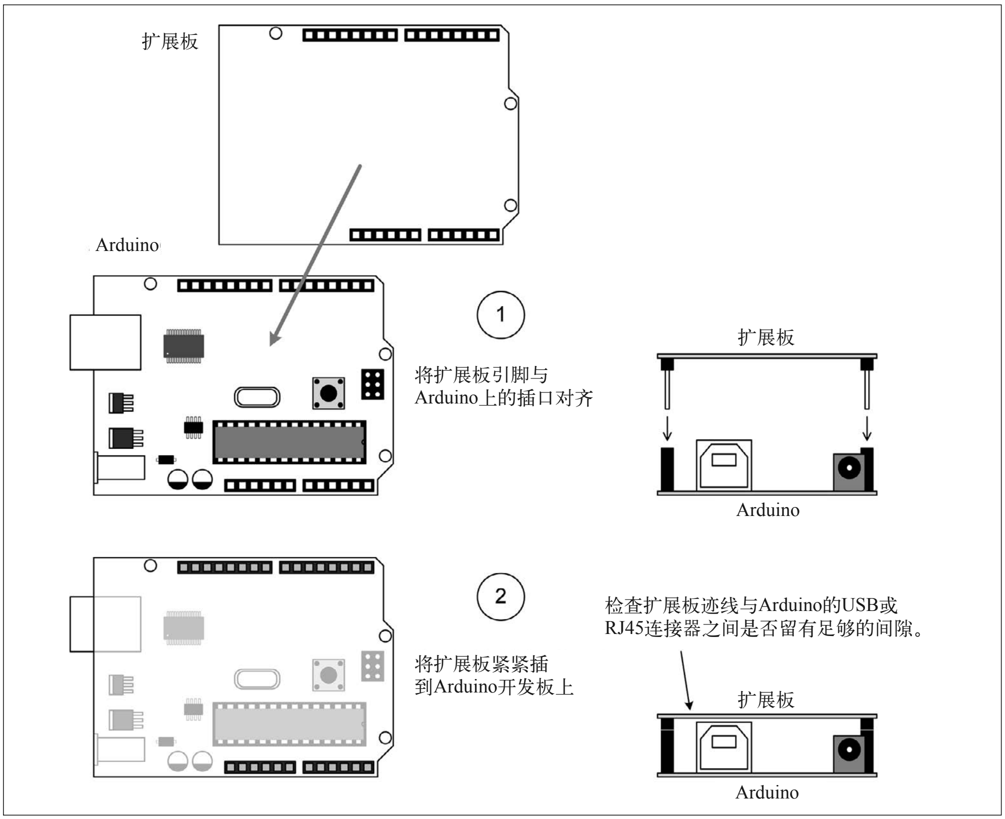


图 8-1：安装于 Arduino 开发板上的扩展板

对于采用 R3 引脚布局的新型 Arduino 开发板，每排引脚插口的末尾有两个引脚插口扩展板不会用到。这并不重要，因为这些额外的插口要么是其他引脚的重复，要么处于未连接状态。就 Mega 开发板来说，其扩展板安装见图 4-20，Mega 开发板上的大部分引脚并未被连接到扩展板，而其他一些则是叠加于其上的扩展板 PCB 无法访问的。所有基线引脚与信号对扩展板都可用。

请务必检查 Arduino 开发板与扩展板 PCB 各个组件之间的间隙。USB 连接器或 RJ45 插口有时可能与扩展板意外连接，导致电路短路。

有时，你可能遇到 Arduino 开发板上的部件与扩展板底部的电路迹线和焊盘发生接触。此时，可以将一小块绝缘胶布或者一些厚卡片纸用作绝缘垫片，将它们阻隔开来。但更好的处理方法是使用间隔器与铆合螺柱，物理隔绝两个板子。间隔器与铆合螺柱是短的金属或尼龙管（长为 7/16 in~1/2 in，或 11 mm~12 mm），带有足够的中心孔洞以安装螺钉（通常是 2-56SAE 类型，或适当的公制单位大小）。铆合螺柱与间隔器的不同是：铆合螺柱带有内部螺纹，而间隔器则没有。使用间隔器或铆合螺柱可以让上下两层板子保持合适的距离，防止因意外接触而发生短路，请参考图 9-3。此外，使用它们也能在两个或多个板子之间形成稳固的机械连接。

当一个 Arduino 开发板上堆叠两个或者多个扩展板时，可以选用带有间隔器的机械螺钉。或者也可以考虑使用某种铆合螺柱，一端是带有螺丝的螺栓，另一端是带有螺丝的螺母。还有一种螺钉（jack screws），它们在 PC 机中常见，如果你自己组装过电脑可能已经见过。图 8-2 显示了几种常用的间隔器与铆合螺柱，它们可以由尼龙、铝、不锈钢、黄铜、塑料制造。

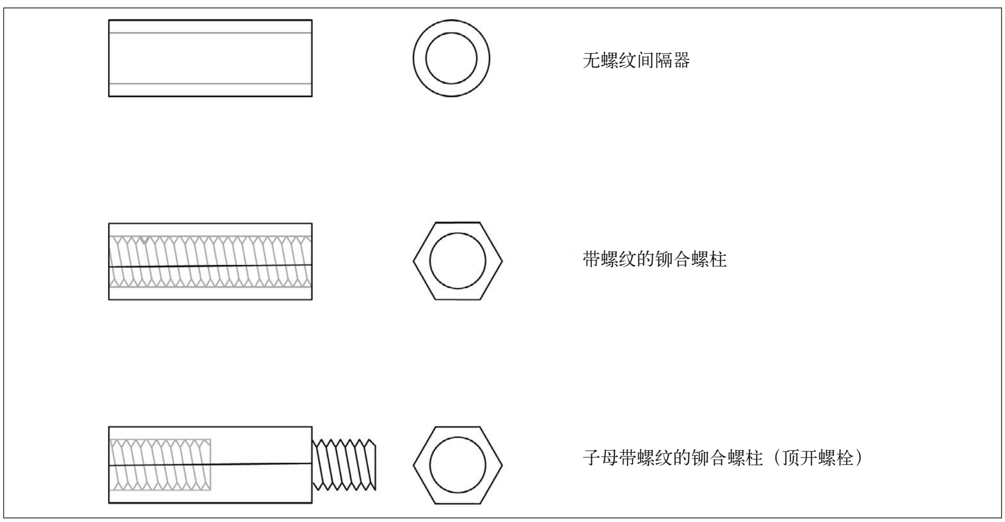


图 8-2：几种间隔器与铆合螺柱

你也可以通过将 PCB 样板削成条制作间隔器，插入上层板子的孔洞中使用。这对于从物理上连接两块板子没什么帮助，但它能提供足够的空间，防止上下两块板子上的元件发生意外接触。

可以从多种渠道购买间隔器或铆合螺柱，比如 Amazon（http://amazon.com）、McMaster-Carr（http://www.mcmaster.com）、Mouser（http://www.mouser.com）、Digi-Key（http://www.digikey.com）。如果你所在的城市正好有一个硬件制造供应商，那么也可以在本地购买它们以及其他有用的组件（比如 1-72 或 2-56 机械螺丝与螺母）。

8.3 堆叠扩展板

有时，采用扩展引脚插口让扩展板适合堆叠毫无意义。比如，如果一个扩展板拥有大量需要垂直通路（vertical access）的连接器，那么有可能没有足够空间让另一个扩展板安装在它上面。然而，有时候你会遇到支持堆叠的扩展板，但没有正确的连接器类型。

可以使用下面两种技术之一，将一个扩展板叠加到另一个扩展板之上：扩展插口引脚，或者偏移引脚与插口。扩展引脚方法允许被堆叠的扩展板沿着垂直方向对齐。偏移引脚与插口的设计会让被堆叠的扩展板发生一定的移动，在引脚行与插口行之间产生一定的偏移量。如果扩展板都以相同方式移动，结果看起来就像“楼梯”一样，并且开发板上用来安装螺钉的孔洞将无法正确对齐。

使用扩展引脚需要引脚插口带有长长的引脚，以便将其安装在 PCB 上。从扩展板的背后或者焊接面伸出的引脚要插入 Arduino 的引脚插口，另一个扩展板可以安装在这个支持堆叠的扩展板之上。引脚与插口偏移技术使用独立的引脚插排将扩展板连接到 Arduino，使用独立的插口插排连接另一个扩展板。它们采用并排方式安装，并且尽可能地靠近，将扩展板之间的偏移降到最小。

8.4 常用Arduino扩展板

本部分将介绍一些常见的扩展板。请注意，它们绝不是一个完整的清单，一些新的扩展板不断涌现，而旧的扩展板不断停产，或者不能再用。在这个快速成型、快速生产、低制造成本的时代，一款扩展板可能很快出现，然后几个月之后就消失不见。为了掌握有哪些扩展板可用，你需要检查 eBay 上的许多中国供应商，以及 Amazon.com、Adafruit、SparkFun、SainSmart 和其他网站的产品目录。请参考附录 C，了解有关 Arduino 扩展板的已知资源。表 8-2 列出了此处提到的扩展板的供应商与制造商。

下面给出本章要介绍的各种扩展板，并将其分门别类地进行整理。

- 输入 / 输出
  - I/O 扩展板
  - I/O 拓展板
  - 继电器扩展板
  - 信号路由扩展板
- 存储器
  - SD 与 microSD 卡闪存
- 通信
  - 串行 I/O
  - 乐器数字接口
  - 以太网
  - 蓝牙
  - USB

    - ZigBee
    - 控制器局域网
- 原型
- 运动控制
    - DC 与步进电机控制
    - PWM 与舵机控制
- 显示器
    - LED 阵列
    - 七段 LED 显示器
    - LCD 显示器
    - 彩色 TFT 显示器
- 仪表扩展板
    - 数据记录
    - 逻辑分析器
    - 24 位模数转换器
    - 12 位数模转换器
- 适配器扩展板
    - Nano 适配器
    - 接线端子适配器
- 混杂扩展板
    - 接线端子 / 原型板
    - 多功能板

此外，本部分还将对一些不常用的扩展板进行介绍，这些扩展板一般应用于特定场合，比如 CNC 雕刻机控制接口、RepRap 控制接口、FPGA 游戏控制器。

8.4.1 输入/输出

输入 / 输出扩展板将 Arduino 的各种 I/O 引脚引出到连接器上，这些连接器比 Arduino 电路板上的引脚更坚固耐用（就 Arduino Nano 来说，开发板下面的引脚被连接到 Nano PCB 的接线端子上）。

I/O 扩展板大致被归类为扩展板（Extension Shields）或拓展板（Expansion Shields），尽管我们常常使用“拓展板”同时指代二者。扩展板只是把 I/O 引脚从 Arduino 上引出来，它不会改变信号，仅仅使用不同类型的连接器。而拓展板则采用有源电子器件，增加离散数字 I/O 通道数量。这些类型的扩展板采用 SPI 或 I2C 与 Arduino 开发板进行通信。

8.4.2 I/O扩展板

这种扩展板将输入 / 输出信号从 AVR 芯片引出到连接器上，这些连接器比 Arduino 开发板上使用的引脚插口要坚固耐用。一些 I/O 扩展板还提供某些类型的缓冲，但大部分只是简单地将信号从 Arduino 开发板引出来而已。有时，它们也被称为“拓展板”，其实这是不对的，因为它们只将现有信号从 Arduino 的一个连接器传送到扩展板上的另一个连接器。

SainSmart 传感器扩展板（http://bit.ly/sainsmart-sensor-shield）

这是一款可堆叠扩展板（注意图 8-3 中的偏移引脚与插口排）。AVR I/O 被引出来，连接到多引脚插口、排针区以及 2 个十针接头，它们适用于带状电缆 IDC 类型的连接器。该扩展板上还有一个重置按钮。SainSmart 传感器扩展板能够与任何一款带有基线引脚配置的 Arduino 开发板配合使用，包括 Mega 开发板。

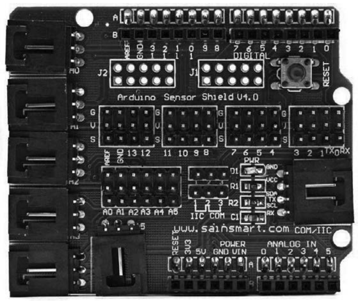


图 8-3：SainSmart I/O 拓展板（传感器接口）

图 8-4 显示了沿着 PCB 边缘分布的大的模块式连接器。它们是多针连接器，有时也称为“带扣”连接器，配合三针与四针插头使用。多线电缆很常见，你可以从多个供应商那里购买，其中之一（除了 SainSmart 之外）是 TrossenRobotics（http://bit.ly/trossen-robotgeek）。

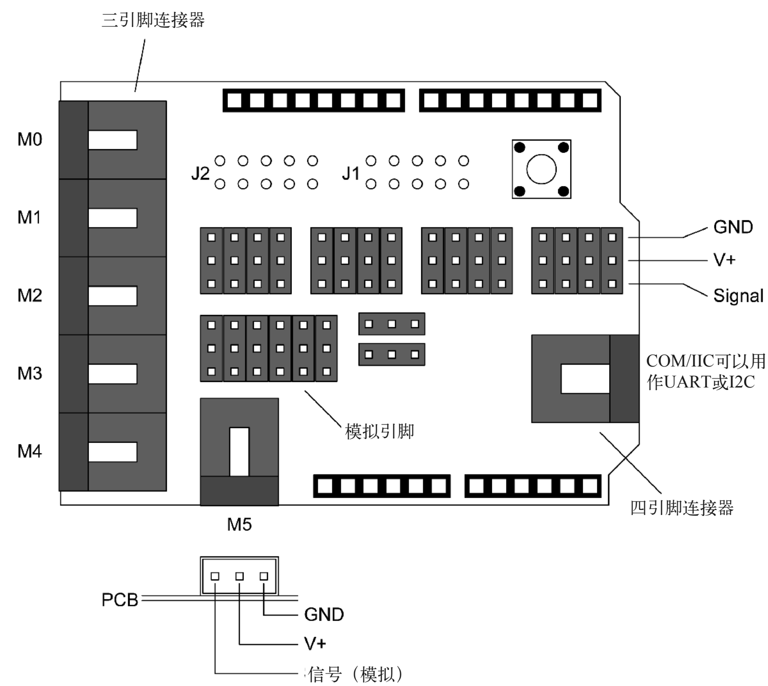


图 8-4：SainSmart I/O 拓展板引脚与连接器布局

TinkerKit 传感器扩展板（http://bit.ly/tinkerkit-sensor）

该扩展板（图 8-5）支持堆叠，采用长引脚插口、12 个三针连接器以及 2 个四针连接器。此外，还有一个重置按钮位于四针连接器之间。

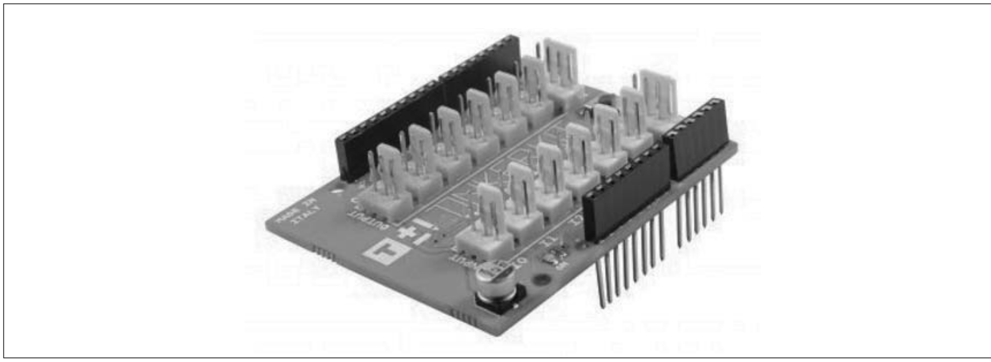


图 8-5：TinkerKit I/O 拓展板

TinkerKit 传感器扩展板起初被设计为，与 TinkerKit 生产的各种传感器和发动机模块一起使用，但你完全可以像使用其他任何 I/O 扩展板一样使用它。图 8-6 显示了 PCB 上各种连接器的布局情况。它们采用三线式方案，和前面 SainSmart 开发板上使用的一样。关于 TinkerKit 模块的更多内容，请参考第 9 章。

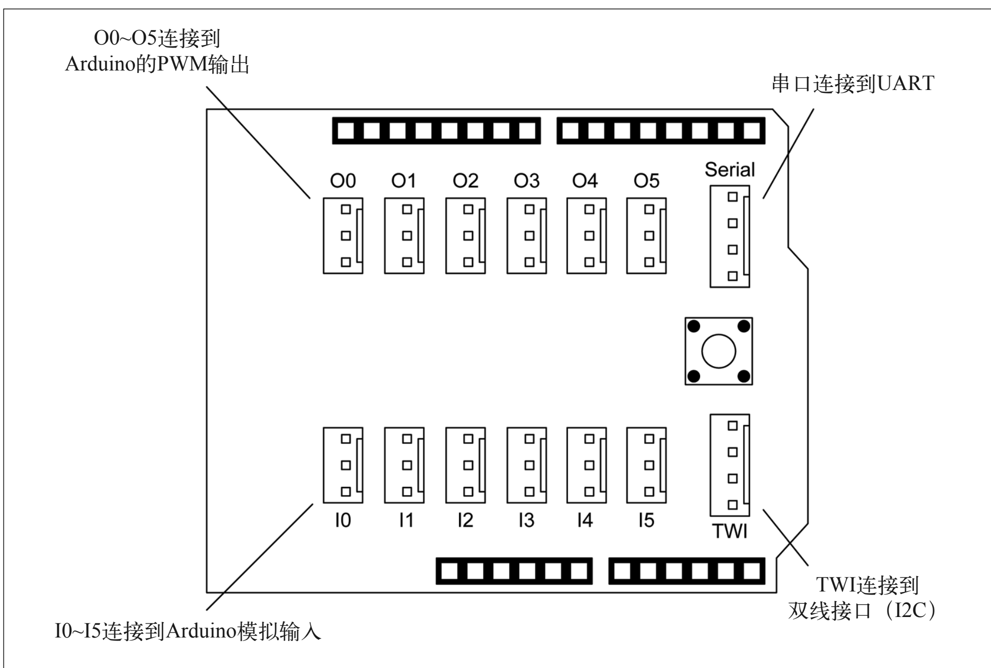


图 8-6：TinkerKit I/O 拓展板连接器

尽管 TinkerKit 目前的处境并不妙，但你仍然可以从 Mouser（http://www.mouser.com）或其他渠道购买到它的产品。也可以从 GitHub（https://github.com/TinkerKit）下载到可用的软件库。

TinkerKit Mega 传感器扩展板（http://bit.ly/tinkerkitmega-sensor）

TinkerKit Mega 传感器扩展板（图 8-7）的设计初衷是，将 Arduino Mega、Mega2560、Mega ADK 开发板额外的 I/O 引脚引出来。它采用长引线引脚插口，支持堆叠功能，在 PCB 上也有一个重置按钮。与上面介绍的 TinkerKit 板相比，它更大一些。

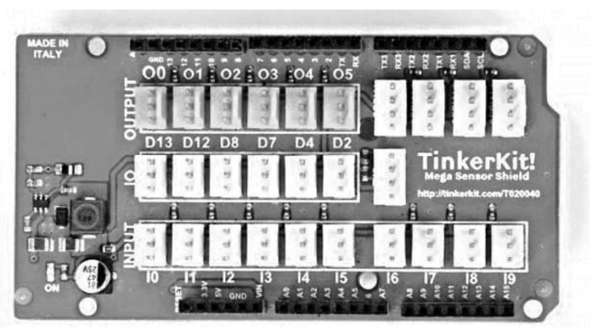


图 8-7：针对 ArduinoMega 开发板的 TinkerKit I/O 拓展板

Grove Base 板

Seeed Studio 销售的 Grove 组件深受 Arduino 用户的喜爱，它由基板与模块组成，提供大量模块供用户选择，基板集成了一块与 Arduino 兼容的 ATMEGA328p MCU。这块板子由 Linaro.com（96Boards.org）设计，实物见图 8-8。

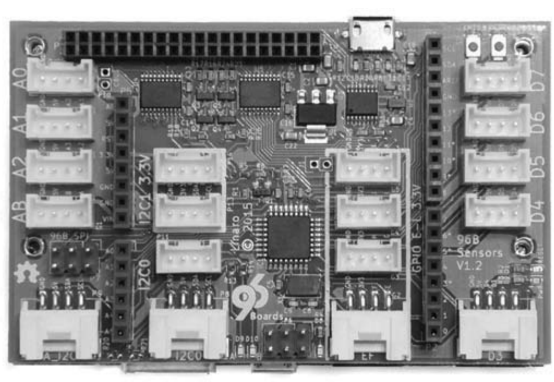


图 8-8：Grove Base 板

Seeed Studio 也销售一块无源（即无板载 MCU）拓展板，它搭载的同样是 Grove 模块系统。虽然目前已经停产，但你仍然可以从一些渠道购买到。无源 Seeed Studio Grove Base 板见图 8-9。

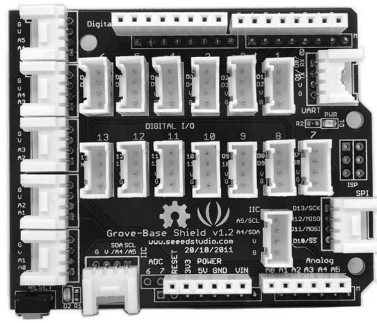


图 8-9：Seeed Studio 无源 Grove Base 板

有关 Grove 系列模块以及兼容接口板的更多信息，请访问 Seeed Studio 维基页面（http://bit.ly/seeed-grove）。

CuteDigi 传感器拓展板（http://bit.ly/cutedigi）

从技术层面讲，它属于 I/O 扩展板，不是一个真正的拓展板。它出自 CuteDigi（图 8-10），使用排针代替连接器，将信号从 Arduino Mega 系列开发板引出来。虽然不支持堆叠功能，但其 PCB 上的引脚呈垂直排列，在其上叠加其他板子毫无意义。PCB 上印制的标签十分清晰，各项功能一目了然。

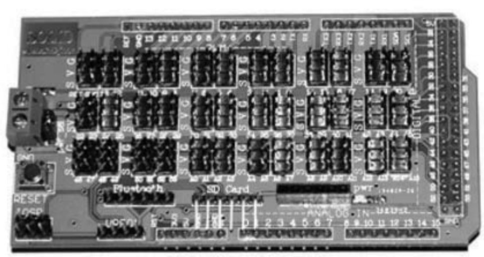


图 8-10：带有 SD 闪存与蓝牙连接的 CuteDigi Mega 扩展板

这块板子的有趣之处在于，它也包含带有直角引脚的针头，用于连接到 SD 类型的闪存卡槽，还有一个排针连接蓝牙模块。连接器采用与其他 I/O 扩展板一样的 S-V-G 方案（信号、V+、地）。

8.4.3　I/O拓展板

不同于前面讲到的 I/O 扩展板，I/O 拓展板提供了额外的 I/O 功能，常见形式是离散的数字 I/O（有些板子的确有模拟能力）。因为这些板子除了带有各种连接器之外，还有有源电路，所以它们要比普通的扩展板贵得多。I/O 拓展板最大的优点是提供多个 I/O 通道，只使用 I2C 或 SPI 连接到底部的 Arduino 开发板上。这样就使得 Arduino 留有更多引脚，提供给其他应用使用。

Macetech Centipede 板（http://bit.ly/macetech-centipede）

Macetech Centipede 板（图 8-11）采用 Arduino I2C 接口，提供 64 个通用的离散型数字 I/O 引脚。这些引脚被划分为 4 组，每组 16 个，每个组都由一个 I2C I/O 扩展芯片控制。

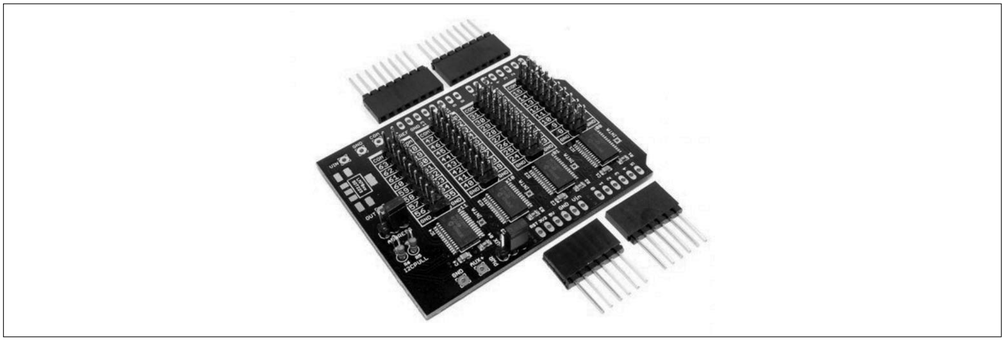


图 8-11：Macetech Centipede I/O 拓展板

图 8-12 显示了 Macetech 板上 I/O 引脚布局的情况。每一个 MUX（多路复用器）IC 控制 16 个引脚，或者一个 I/O 引脚区。请注意各个区域中的引脚是如何进行编号的，在每个引脚的一侧都能找到它对应的数字编号。

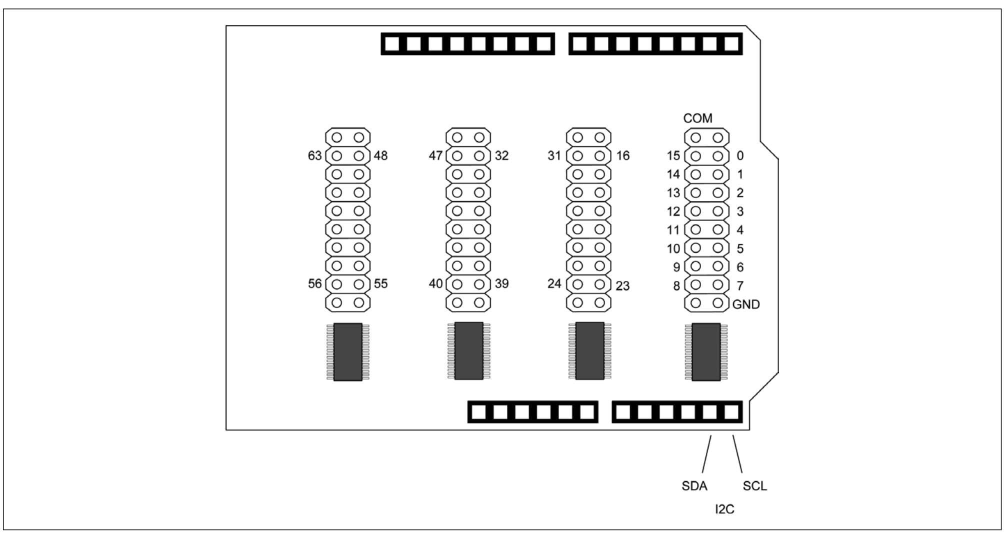


图 8-12：Macetech Centipede I/O 引脚布局

LinkSprite I/O 拓展板（http://bit.ly/linksprite）

这是一款支持堆叠功能的板子，采用 MCP23017 I2C I/O 扩展芯片提供 16 个额外的离散型数字 I/O 引脚（图 8-13）。请注意，这块板子被设计为与 ArduinoR3 系列开发板一起使用，采用 Uno R3 与 Leonardo 开发板扩展连接器上的最后两个引脚（9 号与 10 号，依次为 SDA 与 SCL）。

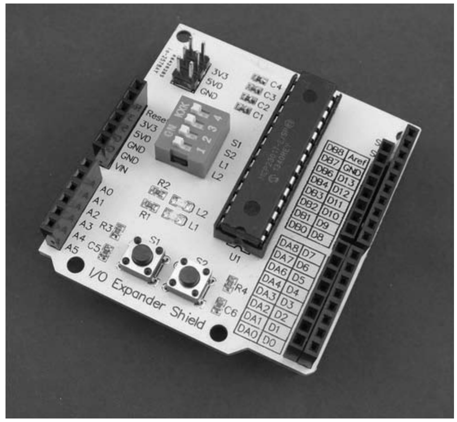


图 8-13：CuteDigi 16- 信道 I/O 拓展板

扩展的 I/O 引脚被排列成两组，每组有 8 个离散信道，分别为 GPIOA 与 GPIOB。如图 8-14 所示，它们紧挨着数字 I/O 引脚插口板。如果有另一个板子叠加于其上，使用这些引脚时可能会遇到困难。使用时，请留心这些小“陷阱”。但如果你只使用这一个板子，或者它位于堆叠的最顶层，则不会出现类似的问题。

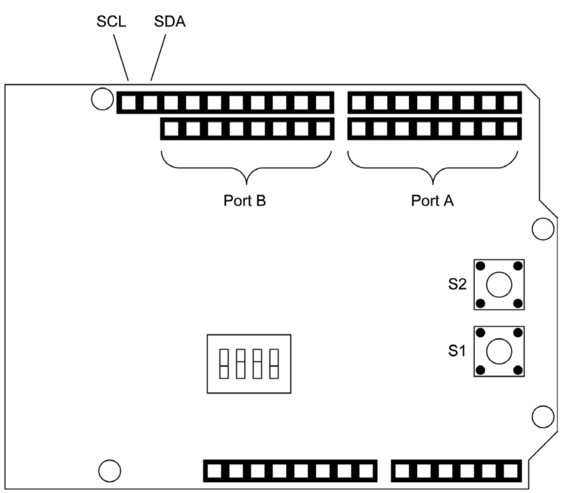


图 8-14：CuteDigi 16- 信道 I/O 拓展板引脚布局

Numato 数字和模拟 IO 拓展板（http://bit.ly/numato-digital-analog）

Numato 推出的数字与模拟拓展板（图 8-15），采用 2 个 MCP23017 I2C 数字 I/O 芯片与一个 NXP 74HC4067 模拟多路复用器 IC（针对模拟信号），提供 28 个额外的离散数字 I/O 信道以及 16 个模拟输入。

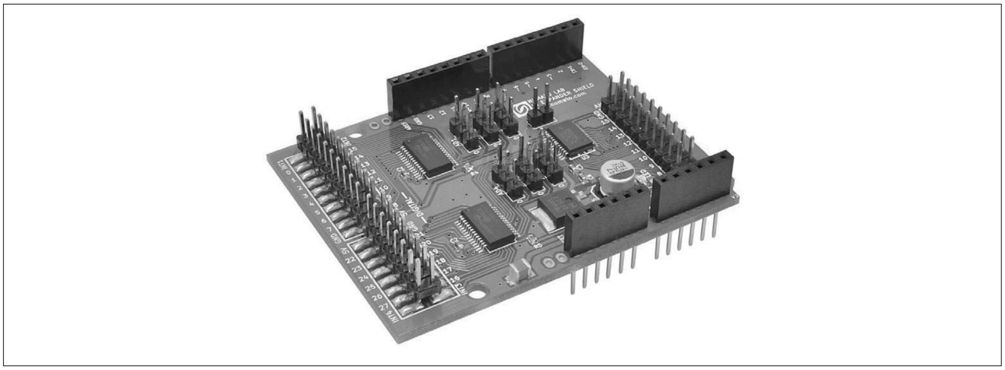


图 8-15：Numato 数字与模拟 I/O 拓展板

如图 8-16 所示，连接 Arduino 的主要接口是 A4 与 A5 引脚上的 I2C 接口。MCP23017 芯片上的中断引脚也被引出到数字 I/O 引脚区。板子中间的 6 个排针用于连接跳线块（jumperblock），选择两个 MCP23017 芯片的 I2C 地址。

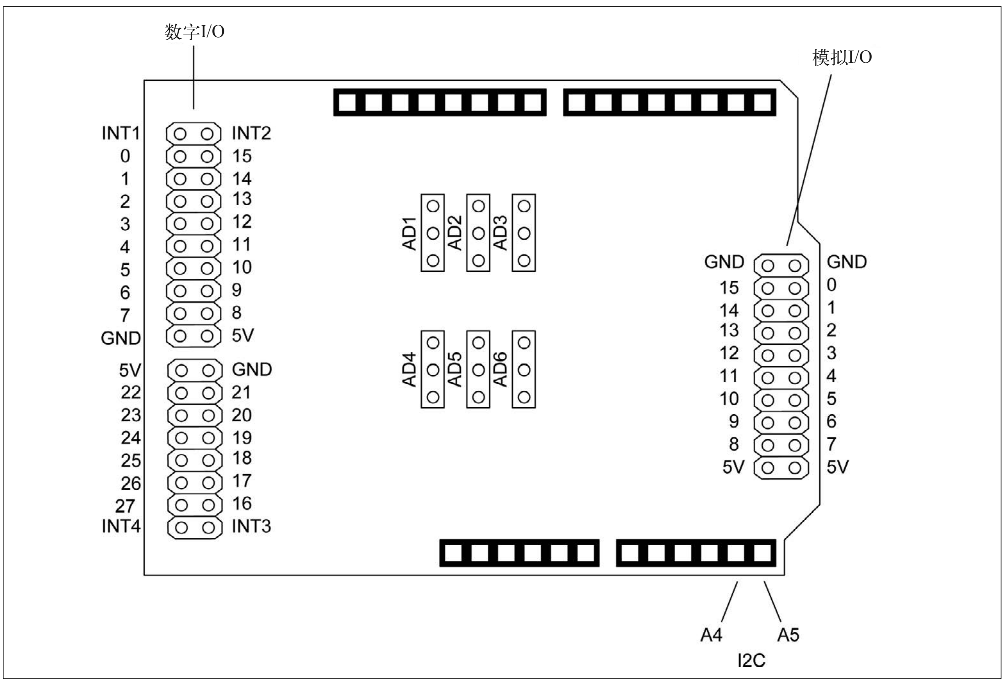


图 8-16：Numato I/O 拓展板引脚分布

8.4.4 继电器扩展板

继电器扩展板一般带有一个或多个继电器模块。这些继电器可以是 5 A 或 10 A 的，你将在下面介绍的各个扩展板中看到它们。此外，干簧继电器采用的是 DIP 封装方式。

评估继电器扩展板时，要注意供应商给出的扩展板的最大额定值。扩展板上的继电器模块可能允许通过 10 A 电流与 120 V（交流）电压，但 PCB 上的连接器与迹线可能承受不了那么大的电流。此外还要注意，并非所有供应商都会降低电流容量以适应 PCB 或连接器的约束。较为妥当的做法是，花一些时间翻阅继电器说明书，供应商的产品图片或电路图中都标出了各个部件的编号，对照说明书查找即可。对于一些扩展板，其上的器件编号可能模糊不清或者被抹去，这时需要你认真观察，并对照说明书理清电路关系（这种情况可能出现在任何扩展板上，而不只是继电器扩展板）。

DFRobot 继电器扩展板（http://bit.ly/dfrobot-relay）

这款扩展板展示了如何充分利用板子的有限空间，使其搭载更大的器件。在 DFRobot 继电器扩展板（图 8-17）中，共有 4 个继电器安装在 PCB 的最边缘。继电器触点负载能力为 3 A 24 VDC 或 120 VAC，最大转换电流为 5 A。它支持堆叠功能，在其上叠加另一个扩展板后，其上的垂直 I/O 引脚可能难以使用。

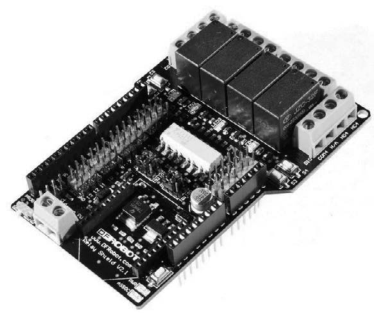


图 8-17：DFRobot 继电器扩展板

该扩展板使用一组跳线，为继电器驱动和 XBee 模块的引脚传送数字信号，见图 8-18。Arduino 的所有数字引脚与模拟引脚都被引出到排针区域。

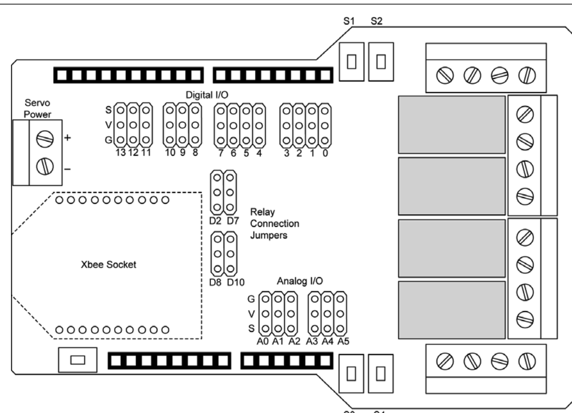


图 8-18：DFRobot 继电器扩展板布局

Numato 继电器扩展板（http://bit.ly/numato-relay）

这块扩展板（图 8-19）采用两个低功耗继电器模块，触点负载能力为 1 A/120 VAC 与 2 A/24 VDC。Numato 扩展板使用 Arduino 的数字引脚 2 与 3。微型端子区块将继电器终端引出。该扩展板支持堆叠功能。

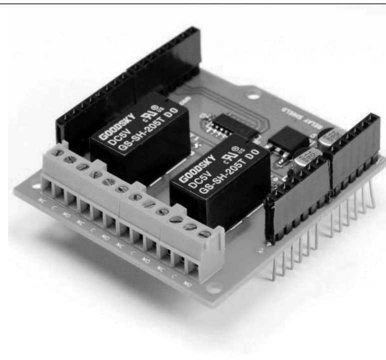


图 8-19：Numato 继电器扩展板

Seeed Studio 继电器扩展板（http://bit.ly/seeedstudio-relay）

Seeed Studio 继电器扩展板（图 8-20）带有 4 个继电器模块，触点负载能力为 10 A/120 VAC。它使用 Arduino 的 4 号 ~7 号数字引脚，每个继电器对应一个引脚。并且每个继电器有一个 LED，用于指示工作状态。该扩展板支持堆叠功能。

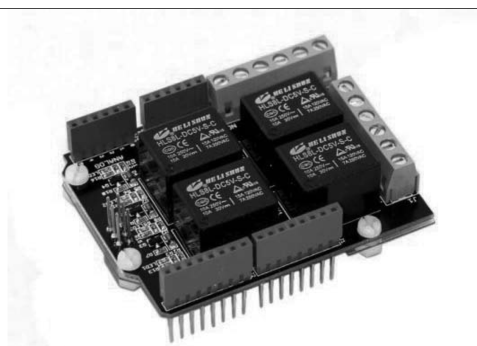


图 8-20：Seeed Studio 继电器扩展板

8.4.5 信号路由扩展板

事实上，并没有很多信号路由扩展板可用，这些信号路由扩展板一般分为两类：被动路由与主动 MUX（多路复用器）路由。

Adafruit Patch 扩展板（http://www.adafruit.com/products/256）

Adafruit 被动 Patch 扩展板（见图 8-21）允许在 4 个 RJ45 型的连接器（亦称 8P8C 连接器）与底部 Arduino 扩展板之间使用插入引脚插口的短贴片线发送信号。

这是一组套件，并不是组装好的扩展板。图片显示的是组装好的扩展板。

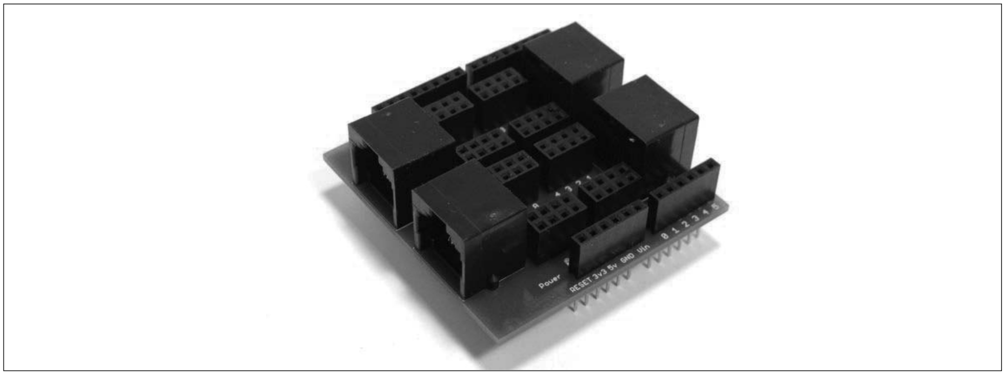


图 8-21：Adafruit patch 扩展板

patch 扩展板背后的主要思想是，将特定信号路由到 Arduino，或者通过传统 Ethernet 线缆从 Arduino 将信号路由到远程连接点，如图 8-22 所示。套件包含 4 个带有“卫星”的 PCB——带有 8P8C（RJ45）插口与引脚，用于连接到免焊面包板上——一个传感器模块或者另一个 Arduino。该扩展板或卫星板上没有有源元件，它只对信号进行路由。

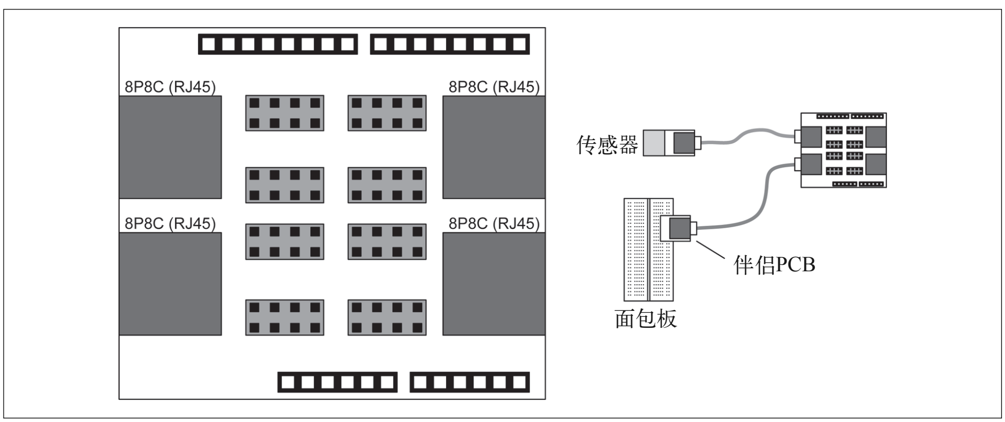


图 8-22：Adafruit patch 扩展板布局与使用

Mayhew Labs Go-Between 扩展板（http://bit.ly/mayhew-gobtw）

Go-Between 扩展板（图 8-23）采用呈矩阵分布的焊点跳线位，将信号从 Arduino 或下位扩展板（lower shield）路由到上位扩展板（upper shield）。如果想堆叠两个扩展板以使用相同引脚实现 I/O 功能，那么使用该扩展板可能会比较方便。如果上位板的引脚能被移动到下位板的不同引脚上而无需任何破坏，将解决很多棘手的问题。只要对软件做较小的改动就能让扩展板正常工作。

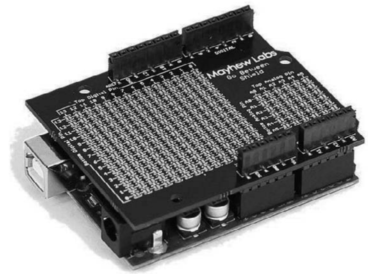


图 8-23：Mayhew Labs Go-Between 扩展板

Mayhew Labs Mux 扩展板 II（http://bit.ly/mayhew-mux）

Mux 扩展板 II（图 8-24）是一块主动扩展板，采用 3 个德州仪器 CD74HC4067 多路复用器芯片与 3 个输出移位寄存器电路，最多可支持 48 个输入或输出。该扩展板使用 4 个 Arduino 数字引脚控制 MUX 芯片与移位寄存器，默认数字引脚是 2、4、6、7。Arduino 的 A0、A1、A2 引脚用作来自 MUX 芯片的输入。

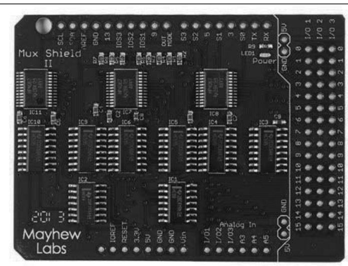


图 8-24：Mayhew Labs 主动信号多路复用扩展板（MuxII）

呈 3×16 排列的焊盘（图 8-25）能够与排针或引脚插口一起使用。每个信道都是双向的，将信号路由到公共输出，或者从公共输入将信号路由进来。每个 MUX 芯片都类似于一排开关，关闭时带有较小的阻抗，打开时阻抗非常高。

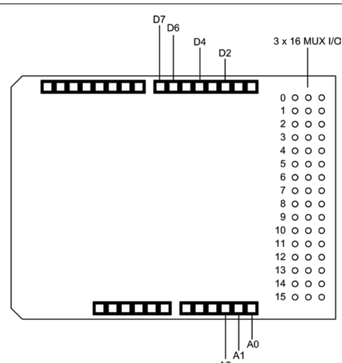


图 8-25：Mayhew Labs 信号多路复用扩展板引脚

Mayhew Labs Mux 扩展板（http://mayhewlabs.com/arduino-mux-shield）

类似于前面介绍的扩展板，这款主动多路复用扩展板采用 3 个 TI CD75HC4067 MUX 芯片（见图 8-26），提供 48 个可编程 I/O 线。同时，它也提供两大块引脚插口头以便访问信号。Arduino 的数字引脚 2~5 提供给 MUX 芯片使用，引脚 A0、A1、A2 是来自 MUX 芯片的模拟输入。该扩展板支持堆叠功能。

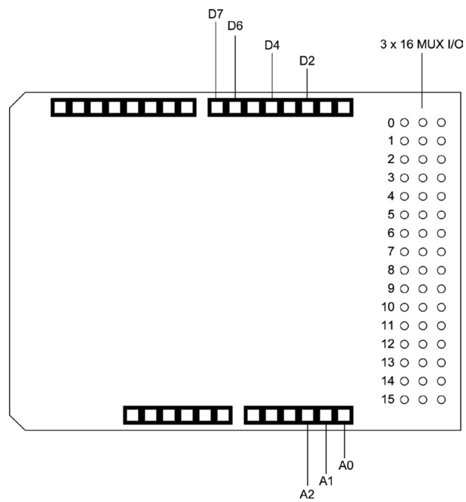


图 8-26：Mayhew Labs 主动信号多路复用扩展板（Mux）

8.4.6 存储器

毫无疑问，SD 与 microSD 闪存卡是为 Arduino 扩充存储的最流行的方式。外部闪存要通过 SPI 接口进行访问。将 SPI 作为主要功能的扩展板（比如 Ethernet、Wi-Fi、USB 主机等）上，常常会看到 SD 或 microSD 插口——作为一项额外的功能。可移动闪存可以很方便地从单独的 Arduino 获取数据，然后将数据导入 PC，并做相应处理。

相关内容讲解中不会使用电路图，主要因为连接到 Arduino 的 SD 或 microSD 接口只是一个 SPI 接口。扩展板在特殊引脚上产生选择信号，这可能导致与现有软件发生冲突。

Seeed Studio SD Card 扩展板（http://bit.ly/seeedstudio-sd-card）

该扩展板（见图 8-27）针对全尺寸 SD 闪存卡设计，它能很容易地与带有适配器的 microSD 卡一起使用。SPI 接口使用 Arduino 数字引脚 D4、D11、D12、D13。此外，该扩展板也把 ICSP、I2C、UART 引脚引出到 PCB 上的连接器，并且支持堆叠功能。

SHD-SD SD 卡扩展板（http://bit.ly/shd-sd-sdcard）

该 SD 扩展板（图 8-28）的特色在于，提供了一小块原型区域，用户可以将自己的电路添加其中。它可以插带有适配器的 microSD 卡，占用 Arduino 上的 D10、D11、D12、D13 引脚，以便通过 SPI 接口与 SD 卡通信。该扩展卡使用来自 Arduino 的 3.3 V DC，比普通扩展板要短一些，带有 Arduino 基线引脚布局，且支持堆叠功能。

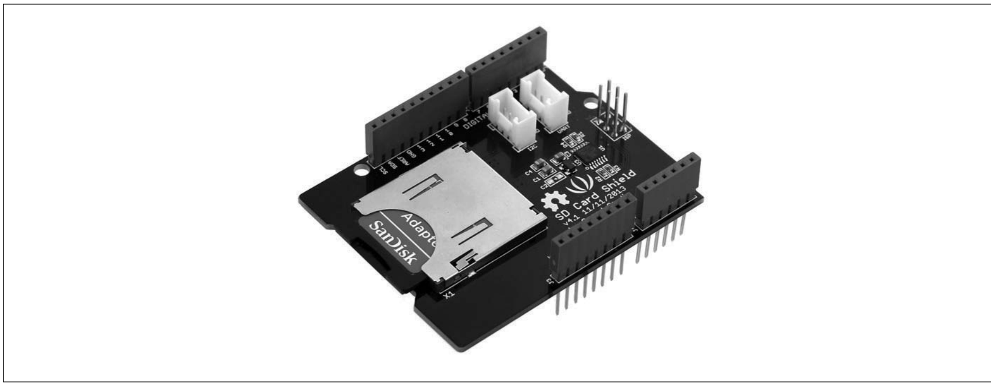


图 8-27：Seeed Studio SD 存储卡扩展板

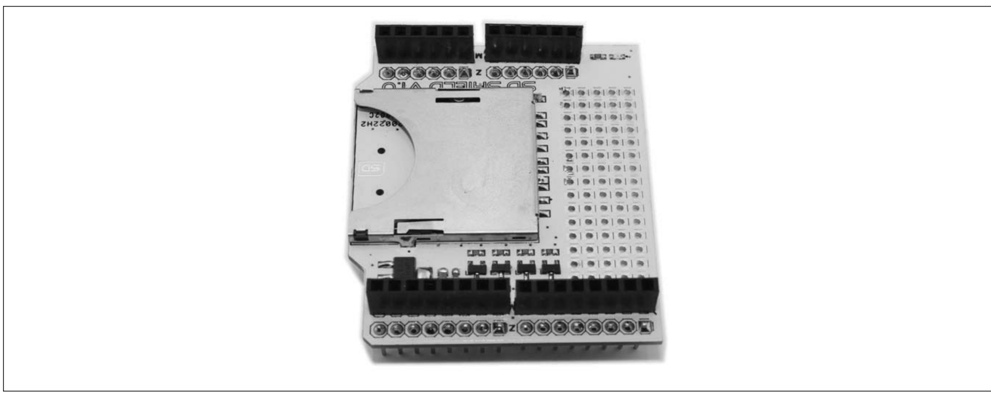


图 8-28：短 SD 卡扩展板

SparkFun microSD 扩展板（http://bit.ly/sparkfun-microsd）

SparkFun microSD 扩展板（图 8-29）只接受 microSD 卡。它包含一个 12×13 的原型区域，不带有引脚插口或排针，但你可以选配它们。该扩展板占用 Arduino 的 D10、D11、D12、D14 引脚。

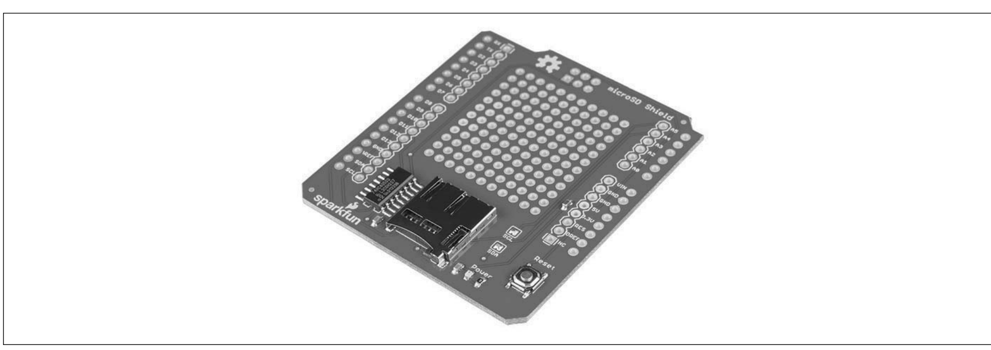


图 8-29：SparkFun microSD 扩展板

8.4.7　通信

尽管 Arduino 本该带有 USB 接口，以便有能力充当串口（从主机系统视角看）或者 Ethernet 插口（在 Arduino Ethernet 上经常看到），但基本的 Arduino（比如 Uno 与 Leonardo）拥有的即插即用式的数据通信接口其实并不多。为此，可以使用电平位移芯片与内置的 UART 或者“位拆裂”技术发送串行数据，但有时，使用连接至 Arduino 的 SPI 接口并让其收发串行数据会更方便。对于其他形式的数据通信，需要有其他外部硬件参与，所以使用现成的扩展板肯定会更容易。

8.4.8　串行I/O与MIDI

尽管 Ethernet 被认为是某种形式的串行数据通信，但此处的“串行 I/O”指的是旧的 RS-232 与 RS-485 标准。虽然它们很旧了，但目前仍然得到广泛应用。旧一点的 PC 有 RS-232 连接器（许多新型机器至少会有一个），而 RS-485 常见于仪表、测试与分布式测量系统。

CuteDigi RS232 扩展板（http://bit.ly/cutedigi-rs232）

这款 RS-232 扩展板（图 8-30）采用一块 MAX232 IC 执行必要的信号电平位移，以收发兼容 RS-232 的信号。它使用一堆跳线配置串口，Arduino 上 D0~D7 的任意两个引脚都可以被用作串口。该扩展板安装有引脚插口，并且支持堆叠功能。

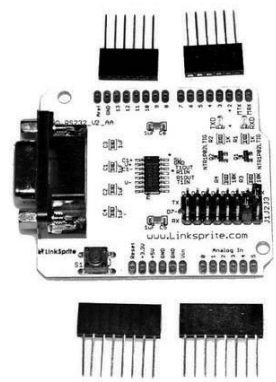


图 8-30：CuteDigi RS232 扩展板

CuteDigi RS-485 扩展板（http://bit.ly/cutedigi-rs485）

CuteDigi RS-485 扩展板（图 8-31）采用 MAX481CSA 芯片提供 RS485 电气接口，它使用 Arduino 的 Rx 与 Tx 端口（依次为 D2 与 D3）。该扩展板 PCB 还提供了一个用于安装 DB-9 连接器的位置，用户可以自己决定是否安装它，并且支持堆叠功能。

SparkFun MIDI 扩展板（http://bit.ly/sparkfun-midi）

MIDI 是一个历史悠久的串行协议，拥有超过 30 年的历史。它用来控制音乐合成器、音序器、鼓机、混音器等。SparkFun MIDI 扩展板（图 8-32）使用 Arduino 的 USART 引脚收发 MIDI 事件消息。

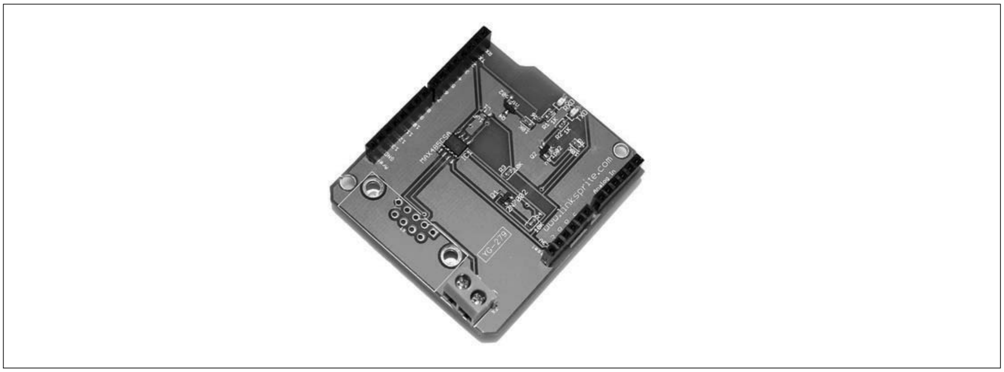


图 8-31：CuteDigi RS485 扩展板

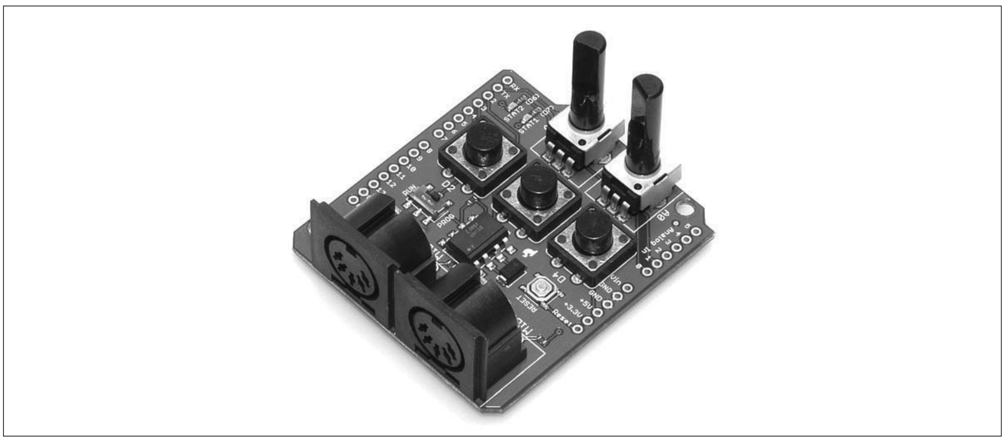


图 8-32：SparkFun MIDI 扩展板

MIDI 扩展板将 D0 与 D1 引脚（依次为 Rx 与 Tx）用作 MIDI 串行 I/O。在 D2、D3、D4 引脚分别对应一个按钮，LED 连接到 D6 与 D7，电位器连接到 A0 与 A1。

8.4.9 Ethernet

Ethernet 扩展板很受欢迎，Arduino IDE 专门为它提供了一个非常全面的 Ethernet 库（更多细节参见第 7 章）。请注意，Arduino 开发板上的 AVR MCU 与扩展板上的 Ethernet 控制器之间使用 SPI 接口进行通信。AVR 不具备 DMA 能力（直接内存访问），在任何情况下，它都没有要直接访问的外存。

Ethernet 扩展板将 SPI 用作与 Arduino 进行通信的接口，AVR MCU 与 Ethernet I/O 芯片之间的数据移动速度存在内在限制，这也体现在 Ethernet 连接的数据移动速度上。

要想在一个以 20 MHZ 运行且使用 SPI 接口的处理器上获取 100 MB/s（100 Base-T）的数据传输速率，根本不可能，获取 10 MB/s（10BASE-T）传输速度也是不可能的。而 5 MB/s 传输速度则是一个更合乎现实的期望。数据仍然通过物理层（真实的 Ethernet）以 10 MB/s 的

速度发送，只不过以字节形式而非连续的流。这完全取决于运行在 AVR 上的软件组装出站数据的速度，以及将其发送到 Ethernet 芯片的速度。所以，你可以用它制作一个 Web 服务器，并将其放入小的外壳之中，比如旧的薄荷糖铁盒。它的速度不会很快，也不能同时处理大量连接。

Ethernet 接口真正的用武之地是，将其用作遥感或控制系统的终端节点。可以将其附加到互联网上，对密码进行某种保护，或者从远程的某个位置获取数据。它也可以用在工业环境（比如工厂）中，将数据提交给中心控制器，或者为分布式 HVAC（采暖、通风、空调）系统控制器（第 12 章）测量温度、湿度以及其他参数。

带有 microSD 卡读卡器的 Vetco Ethernet 扩展板（http://bit.ly/vetco-ethernet）

来自 Vetco 的这款 Ethernet 扩展板（图 8-33）也包含一个 microSD 卡的载卡器和一个重置按钮。Arduino 的数字引脚 D10、D11、D12、D13 被 SPI 接口占用，WIZnet W5100 Ethernet 芯片与 microSD 卡座会使用 SPI 接口。该扩展板支持堆叠功能。

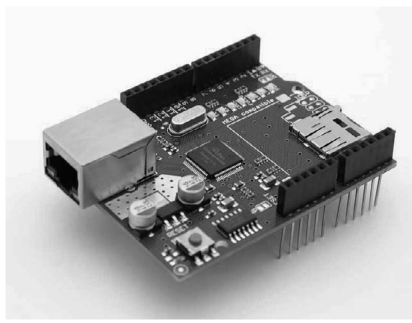


图 8-33：Vetco Ethernet 扩展板

带有 microSD 连接器的 Arduino Ethernet 扩展板 R3（http://bit.ly/ethernet-r3）

该 Ethernet 扩展板来自 Arduino 官方，带有一个 microSD 读卡器、一个重置按钮，以及用于实现 Ethernet 接口（见图 8-34）的所有必需的电子器件。它把数字引脚 D10、D11、D12、D13 用作 SPI 接口，且支持堆叠功能。

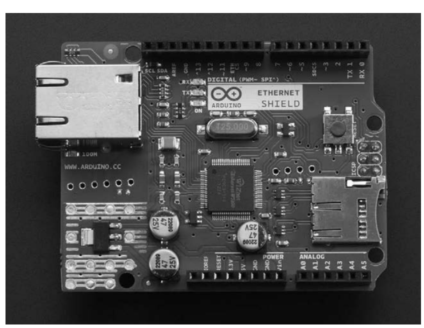


图 8-34：Arduino Ethernet 开发板

8.4.10 蓝牙

蓝牙是一种低功耗、短距离的无线通信技术，原本用于取代计算机与外部设备（比如打印机、键盘、鼠标等）之间的连接线缆。虽然现在也将蓝牙技术用在这些用途之中，但在其他类型的通信应用中也能看到它的身影。市面上有大量蓝牙扩展板可用。

Bluetooth 扩展板（http://bit.ly/dx-bluetooth）

该扩展板带有一个蓝牙模块，且已经安装在 PCB 上。请注意，它支持堆叠功能，但要比标准扩展板短一些。天线是呈现为金色图案，位于蓝牙模块末端（见图 8-35）。该扩展板使用 Rx 与 Tx 引脚（依次为 D2 与 D3）。

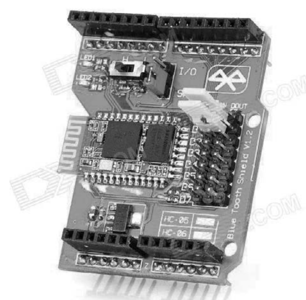


图 8-35：DealeXtreme 蓝牙扩展板

Seeed Studio 蓝牙扩展板（http://bit.ly/seeed-bluetooth）

这款袖珍型可堆叠式蓝牙扩展板（图 8-36）占用 Arduino 的 Rx 与 Tx 引脚。它也把模拟与数字信号引脚引出，与传感器模块连接在一起。

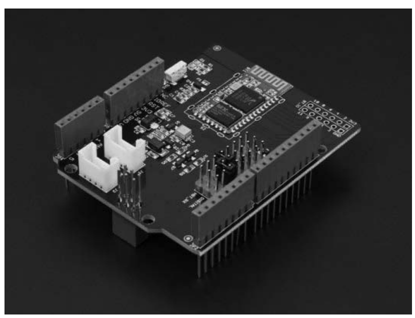


图 8-36：来自 Seeed Studio 的小型蓝牙扩展板

ITEAD Bluetooth Wireless BT Module 扩展板套装（http://bit.ly/itead-bluetooth）

该扩展板（图 8-37）采用标准的蓝牙模块，且在 PCB 上带有一块原型区域。与 Arduino 开发板相比，它属性一个“短”扩展板，无法覆盖整个 Arduino 板。该扩展板还支持堆叠功能。

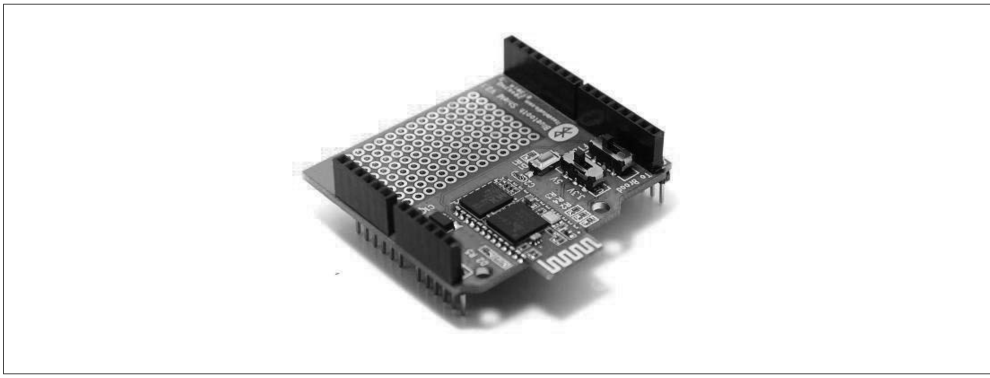


图 8-37：带有原型区域的 ITEAD 蓝牙扩展板

DFRobot Gravity:IO 扩展板（http://bit.ly/dfrobot-gravityio）

DFRobot 多功能扩展板（见图 8-38）将 I/O 扩展能力与一系列蓝牙或 ZigBee 无线模块的引脚插口集成在一起。该扩展板不支持堆叠功能。

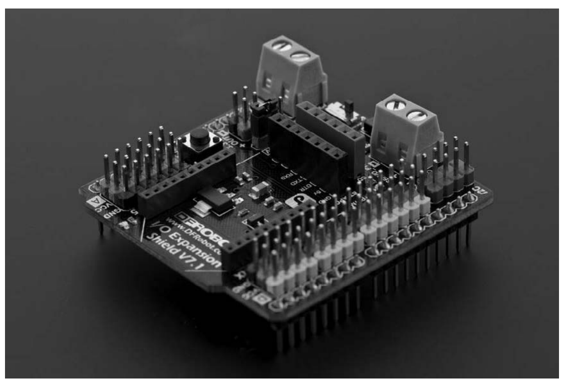


图 8-38：带有蓝牙的 DFRobot 多功能扩展板

8.4.11 USB

8 位 Arduino 不能充当其他 USB 设备的 USB 主机。USB 主机扩展板允许将 USB 设备连接到 Arduino，常见的 USB 设备有键盘、打印机、一些测试仪器以及各种玩具。

ITEAD USB 主机扩展板（http://bit.ly/itead-usb）

该扩展板支持堆叠功能，带有 USB 主机功能。并且，它为 Arduino 数字与模拟信号提供了引脚，还为连接器或排针在 PCB 上提供 2 个十针位（见图 8-39）。该扩展板基于 MAX3421E 芯片，采用 SPI 接口连接到 Arduino。

Circuits@Home USB 主机扩展板（http://bit.ly/circuitsathome-usb）

该扩展板（图 8-40）支持 USB 2.0 全速运行，采用 SPI 接口连接到 Arduino。来自

Arduino 的数字与模拟信号在扩展板 PCB 上可用，借助正确的引脚插口，可以支持堆叠功能。它采用 MAX3421E 芯片，通过 SPI 接口连接 Arduino，占用 D10、D11、D12、D13 引脚。请注意，该扩展板本身不带引脚插口与排针。

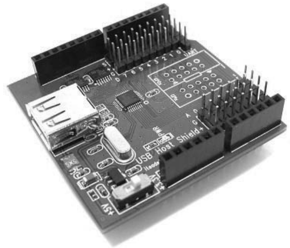


图 8-39：带有 I/O 连接的 USB 主机扩展板

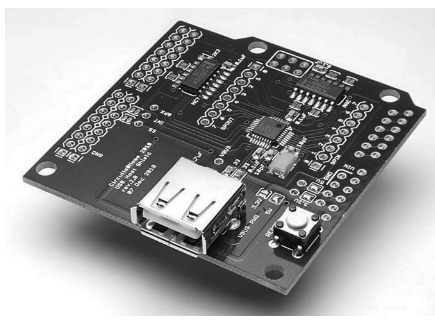


图 8-40：Circuits@Home USB 主机扩展板

Arduino USB 主机扩展板（http://bit.ly/arduino-usb）

类似于其他 USB 主机扩展板，该扩展板使用 MAX3421E 芯片，通过 SPI 接口连接到 Arduino，占用 D10、D11、D12、D13 引脚。三脚或四脚连接器将输入与输出端口引出，这些端口将直接与 TinkerKit 模块一起工作（第 9 章讲解 TinkerKit 模块）。该扩展板支持堆叠功能（见图 8-41）。

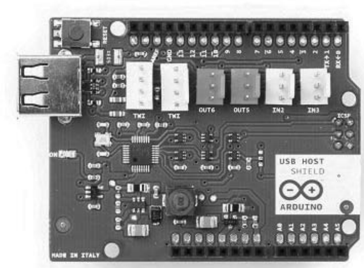


图 8-41：带有 I/O 连接器的 Arduino USB 主机扩展板

8.4.12　ZigBee

ZigBee 是一个广受欢迎的低功耗无线协议。许多 ZigBee Arduino 扩展板使用容易获得的 XBee 模块，但大部分扩展板会容纳带有正确引脚的 RF 模块。有些带有 XBee 模块，有些则不带。一个 1 mW 的 XBee 模块大约 25 美元。

Arduino Wireless SD 扩展板（http://bit.ly/arduino-wireless）

这种 ZigBee 扩展板（图 8-42）上提供了 2 个直插式引脚插口，给 Digi XBee 模块使用，或者任何带有可兼容引脚排列的模块。该扩展板使用 Arduino 的 D4 引脚用作选择，将 D11、D12、D13 引脚用作 SPI 通信。扩展板上的 microSD 插槽也使用 SPI 接口。

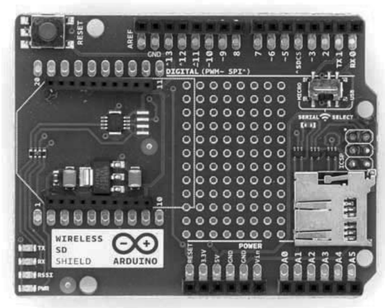


图 8-42：Arduino ZigBee 扩展板

SainSmart XBee 扩展板（http://bit.ly/sainsm-xbee）

该扩展板不带有 XBee 模块，但预留了相应引脚，方便安装标准的 XBee 模块，或任何带有兼容引脚的模块。请注意，它也没有 microSD 插槽，而支持偏移堆叠功能（注意观察图 8-43 中排针与引脚插口的位置）。



图 8-43：SainSmart 小型 ZigBee 扩展板

Seeed Studio XBee 扩展板（http://bit.ly/seeedst-xbee）

该 XBee 扩展板来自 Seeed Studio（图 8-44），针对常见的 XBee 模块预留有安装位置，也提供原型区域。这个扩展板使用 Arduino 的 Rx 与 Tx 引脚，有一片跳线引脚将 Arduino 的 Rx 与 Tx 信号路由到无线模块。

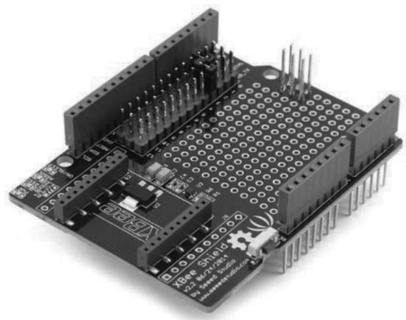


图 8-44：带有原型区域的 SainSmart ZigBee 扩展板

8.4.13 CAN

与车辆、工业领域、一些军事设备中常见的 RS-485 类似，控制器局域网（CAN，亦称作 CAN 总线）也采用差分信号技术。它的传输速度相当快，高达 1 MB/s，同时具备信号冲突检测与错误检测功能，支持多节点接入。它经常应用于旧车的 OBD-II 车载诊断系统、电动车，以及科学仪器的分布式传感器中，甚至被集成到高档自行车中。

Seeed Studio CAN-BUS 扩展板（http://bit.ly/seeed-canbus）

Seeed Studio 推出的 CAN 接口扩展板（图 8-45）采用一个带有 SPI 接口的 MCP2515 CAN 总线控制器与 MCP2551 CAN 收发器芯片。并且，为 CAN 总线信号提供端子台与 DB-9 连接器。此外，该扩展板还将 I2C 与 UART 通信从 Arduino 上引出，其引脚布局如图 8-46 所示。



图 8-45：带有辅助 I/O 连接器的 Seeed Studio CAN 扩展板

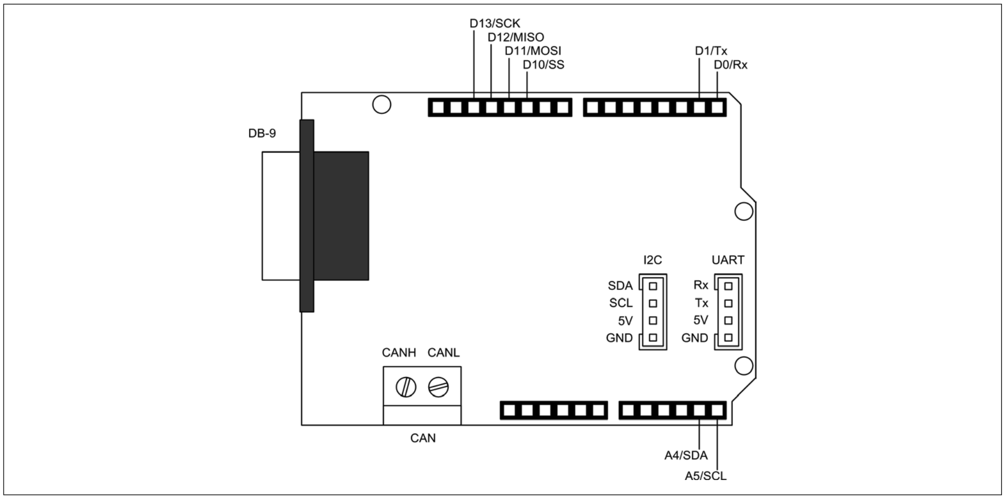


图 8-46：带有辅助 I/O 引脚布局的 Seeed Studio CAN 扩展板

SparkFun CAN-BUS 扩展板（http://bit.ly/sparkfun-canbus）

SparkFun 推出的 CAN 扩展板（图 8-47）集成了许多有用的特性，你可以使用它们创建读取 OBD-II 以及捕获数据的设备。它为 CAN 总线信号提供 DB-9 连接器，也通过一个四脚排针提供信号。同时，还为外部 LCD 显示器提供了连接点，还包含一个 EM406 GPS 模块。

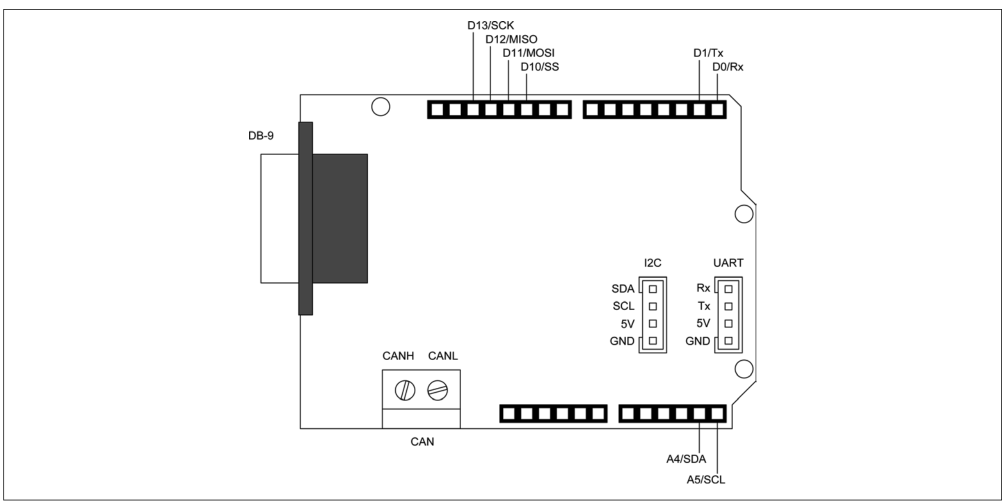


图 8-47：带有 microSD 卡槽与数字摇杆的 SparkFun CAN 扩展板

有趣的是，它还提供了一个 4 位二进制摇杆（four-position binary joystick），集成了一个 microSD 闪存卡槽。摇杆连接到 Arduino 的模拟输入上。通过数字引脚 D9 与 D10 分别选择 CAN 芯片与 SD 闪存。该扩展板不使用引脚 D3、D4、D5 与 A0，其引脚布局如图 8-48 所示。

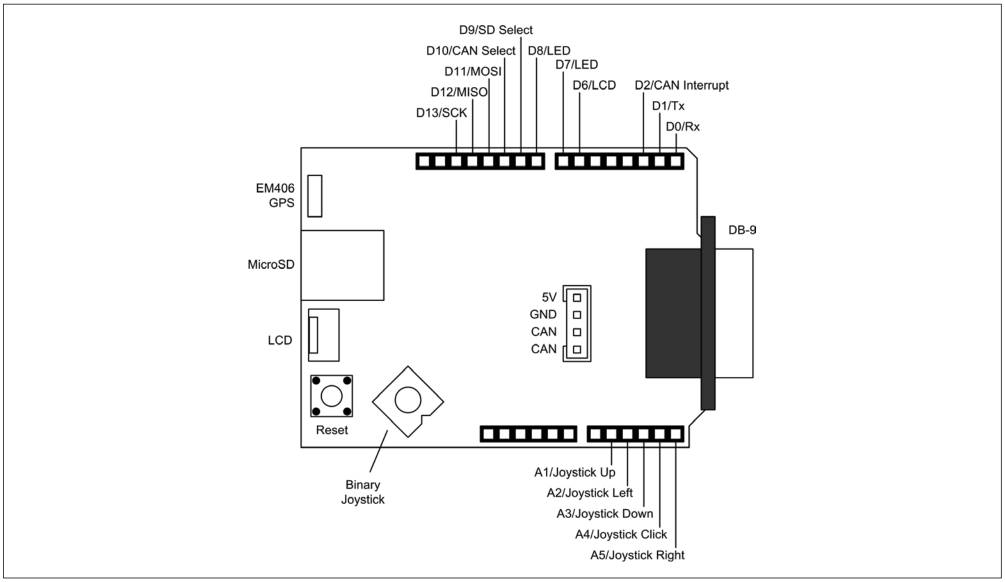


图 8-48：SparkFun CAN 扩展板 I/O 引脚布局

LinkSprite CAN-BUS 扩展板（http://bit.ly/linksprite-can-bus）

从物理外形看，LinkSprite CAN 扩展板（图 8-49）类似于 Seeed 扩展板。它为 CAN 信号提供了一个 DB-9 连接器与一个双位端子台。SPI 接口使用引脚 D10、D11、D12、D13 连接到 MCP2515 CAN 芯片。

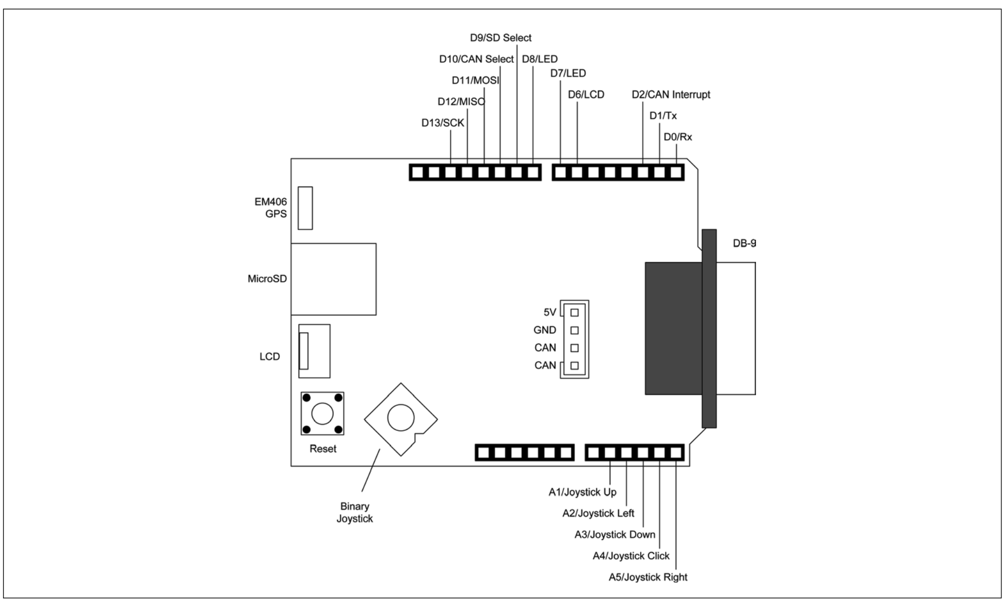


图 8-49：LinkSprite CAN 开发板

8.4.14 原型

如果想制作自己的扩展板，你可以使用原型扩展板搭建原型（甚至是永久扩展板）。这与第 10 章中介绍的创建扩展板的过程不一样，那涉及 PCB 的设计以便实现大批量生产。此处要讲的原型扩展板如图 8-50 所示，它安装有一个温度传感器以及继电器。一个电位器连接到下面 Arduino 开发板的 +5 V、地与 A0（模拟输入 0）引脚上。可调电阻控制温度设置点。

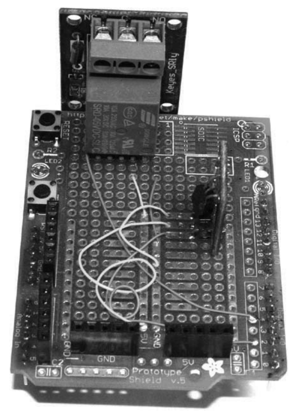


图 8-50：原型温度传感器 / 控制器扩展板

该原型控制一台老式（且非常危险的）便携式电暖器，它使用双金属温度控制器，温度控制范围大约为 +/−15°。由于继电器触点负载能力仅为 10 A/120 VAC，而加热元件为 15 A，所以它与 24 VAC 的变压器一起使用，以控制 20 A 的接触器。这台电暖器工作得相当好，让我的办公室在整个冬天都比较舒服。为了安全起见，我打算再为它加一个倾斜传感器、输出温度传感器，以及风扇探测器。

本部分所讲的扩展板都很具代表性，没有什么特别复杂的内容。使用 Arduino 引脚做什么完全取决于你的想法（毕竟它是一个原型），所以不怎么需要使用辅助图。

Adafruit Stackable R3 原型扩展板（http://bit.ly/stackable-r3）

该扩展板（图 8-51）以套件形式推出，由一块裸板（bare PCB）、一包元件组成。把它们组装起来并不难，但需要具备一点焊接技术。

Adafruit Mega 原型扩展板（http://bit.ly/ada-mega-proto）

该原型扩展板套件同样来自 Adafruit，它支持堆叠功能，其所有组成元件如图 8-52 所示。请注意，沿着 PCB 边缘的双排连接焊盘允许焊接短距引脚插口，以便轻松访问来自底部 ArduinoMega 开发板的信号。

CuteDigi Assembled Protoshield for Arduino（http://bit.ly/cutedigi-assembled）

该原型扩展板（图 8-53）是已经组装好的成品。PCB 上带有引脚插口，用于访问 Arduino 信号。它不支持堆叠功能。

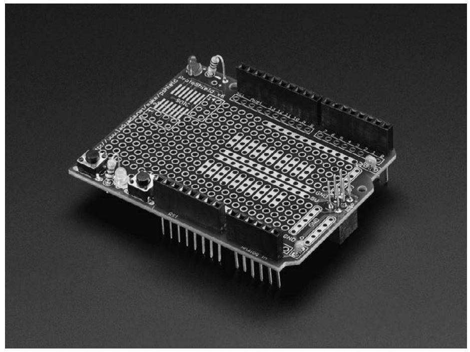


图 8-51：Adafruit stackable 原型扩展板套件

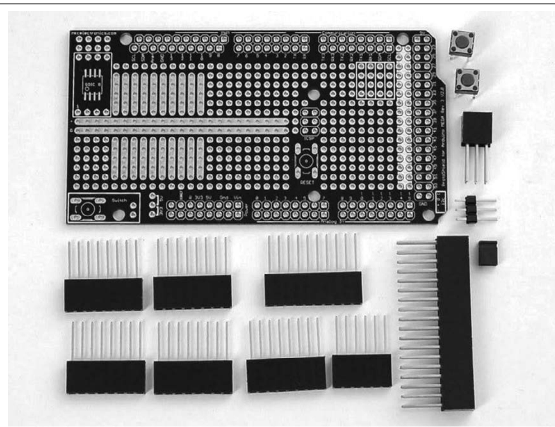


图 8-52：Adafruit Mega 原型扩展板套装

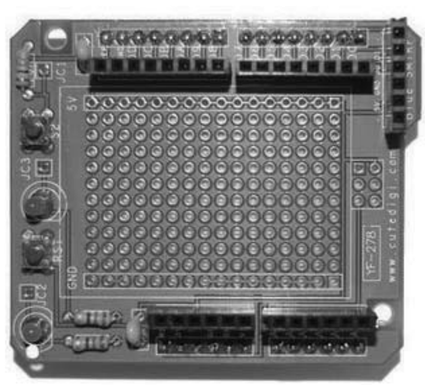


图 8-53：CuteDigi 原型扩展板

CuteDigi Assembled Protoshield for Arduino MEGA（http://bit.ly/assembled-mega）

该扩展板（图 8-54）被设计为与 Arduino Mega 开发板一起工作，它为小外形集成电路（SOIC）提供了一个安装位置，不支持堆叠功能。

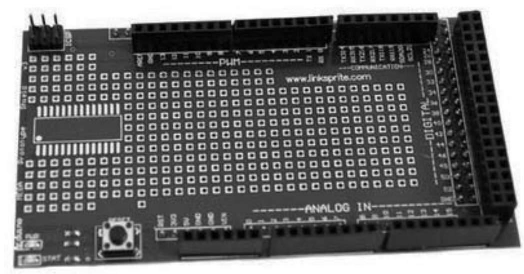


图 8-54：CuteDigi Mega 原型扩展板

CuteDigi Protoshield for Arduino with Mini Breadboard（http://bit.ly/proto-bboard）

该扩展板（见图 8-55）包含一个小的免焊面包板，方便用户搭建自己的电路。

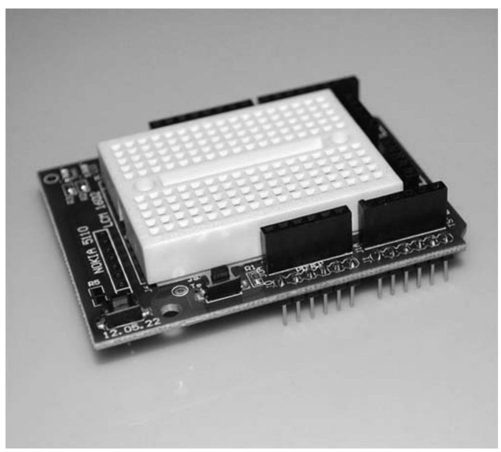


图 8-55：带有面包板的 CuteDigi 原型扩展板

8.4.15　制作自定义原型扩展板

你可以快速制作一个可正常工作的扩展板，需要的只是一个原型 PCB、一些引脚和一些插口连接器。PCB 的大小其实无关紧要，只要引脚与 Arduino 开发板上的插口对齐即可。

Adafruit DIY 扩展板套件（http://bit.ly/ada-diy）

如果要颁发一个“最简单扩展板套件奖”，那么这个扩展板肯定名列前茅。它由一个原型 PCB 与 4 个长距引脚插口连接器组成（图 8-56），允许在其上添加任何想要的元件。借助该扩展板套件，你可以很容易地设计一个新的原型扩展板，或者将一些元件快速拼

集在一起。更棒的是，你可以灵活使用某些现有模块，而这些模块原来从未打算要连接到 Arduino 上。

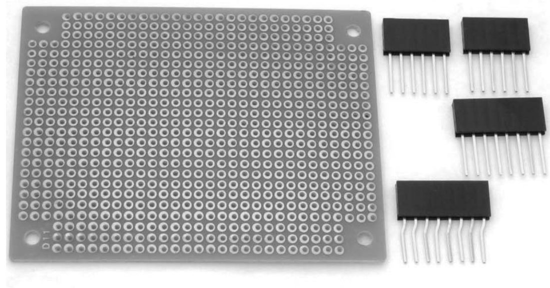


图 8-56：Adafruit DIY 扩展板套件

不过，令人遗憾的是，Adafruit 已经不再生产该产品，但你可以自己动手制作。需要的全部元件包括一个原型 PCB（可以从多个渠道购买）和引脚插口，这些都可以在 Adafruit 以及其他供应商处购得。由于 Arduino 开发板上的引脚焊盘采用的是工业标准 0.1 in（2.54 mm）间距，所以用基本的原型扩展板很容易制作出自己的电子作品。

8.4.16　运动控制

运动控制是 Arduino 应用比较广泛且令人感兴趣的领域。从可编程移动机器人到 CNC 雕刻机，再到 3D 打印、激光扫描，乃至太阳能面板的太阳自动跟踪控制器、动态雕塑，都能见到 Arduino 的身影，它从一开始就被用于控制 DC 发动机、舵机、步进电机。如你所料，市面上有大量扩展板可以应用于每种发动机，此处介绍的只是其中很少一部分。

8.4.17　DC与步进电机控制

通常，基于 H-bridge（一种固态电流导向开关）的电机控制器扩展板用于控制有刷 DC 电机或步进电机。这些类型的扩展板基本可以用于控制任意感应 DC 负载，包括螺线管与继电器。

Rugged Motor Driver 扩展板（http://bit.ly/rugged_motor）

Rugged Motor Driver 扩展板（图 8-57）由 Rugged Circuits 推出，它能驱动 2 个有刷 DC 电机，或者一个双极步进电机。其最大工作电压为 30 V，峰值电流为 2.8 A。该扩展板使用 D3、D11、D12、D13 引脚作为使能输入与方向控制输入。使能输入可以由一个 PWM 信号驱动，用于实现对 DC 电机的平滑控制。关于电流处理与软件的更多细节，请前往官网查阅相关资料。

SainSmart Motor Drive 扩展板（http://bit.ly/sainsm-motor）

SainSmart 电机扩展板（图 8-58）基于 L293D 四通道驱动器 IC，它可以驱动 4 个有刷 DC 电机或 2 个步进电机，最大耐压 10 V。并且为外部电力供应提供了接线端子。关于电流处理与软件的更多细节，请前往官网查阅相关资料。

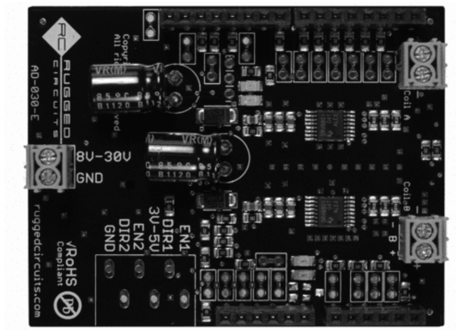


图 8-57：Rugged Motor Driver 扩展板

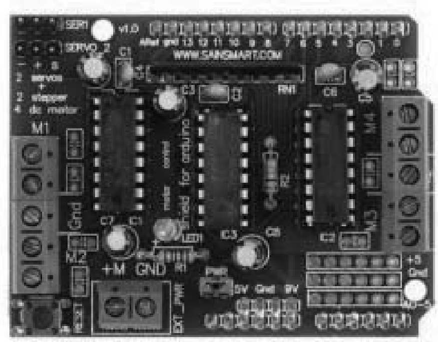


图 8-58：SainSmart L239D Motor Drive 扩展板

Arduino Motor 扩展板（http://bit.ly/arduino-motor）

这款电机扩展板（图 8-59）由 Arduino.cc 出品，基于 L298 双路驱动 IC。它能与继电器、螺线管、DC 电机、步进电机一起使用。其有趣之处在于，它具备测量电流消耗的能力，借助这一特征可以很容易检测出电机是否停转。此外，还要注意它带有模块连接器，可以很方便地连接各种 TinkerKit 模块（第 9 章）。

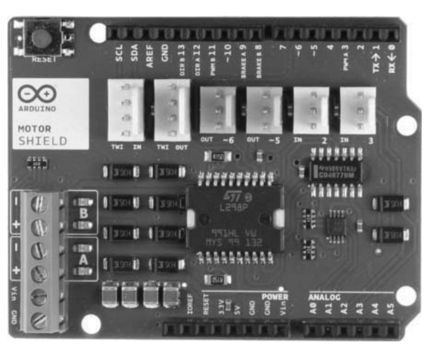


图 8-59：Arduino 电机扩展板

8.4.18 PWM与舵机控制

遥控模型（RC models）与小型机器人中使用的舵机通过定位电枢工作，它相对于一系列宽度不同但频率稳定的控制脉冲。脉冲宽度（on）控制舵机旋转的角度。PWM/ 舵机扩展板能驱动一个 DC 电机，精确控制一个或多个 LED 灯的亮度，或者操作线性驱动器。

16 通道 12 位 PWM/Servo 扩展板（http://bit.ly/ada-16-pwm）

该扩展板（图 8-60）采用一个 PCA9685 16 通道 PWM 控制器 IC，带有 I2C 接口。它有能力针对每个输出产生唯一的可编程 PWM 信号，而且并不要求 Arduino“持续关注”它。

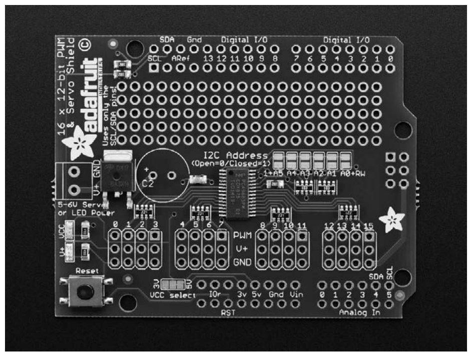


图 8-60：带有 I2C 接口的 Adafruit PWM/ 舵机扩展板

LinkSprite 27 通道 PWM 舵机扩展板（http://bit.ly/linksprite-27-pwm）

该扩展板（图 8-61）由 LinkSprite 推出，采用 STM32F103C8T6 微控制器 IC 产生多达 27 路的独立 PWM 输出。值得注意的是，STM32F103C8T6 是一个 ARM Cortex-M3 32 位 RISC 设备，带有多达 128 kB 的闪存与 20 kB 的 SRAM。实际上，该扩展板上的微控制器比 Arduino 开发板上的 AVR 拥有更强大的计算能力。它通过 SPI 接口与 Arduino 进行通信。

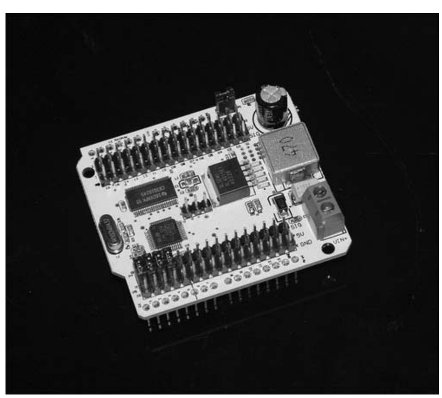


图 8-61：LinkSprite 舵机扩展板

SparkFun PWM 扩展板（http://bit.ly/sparkfun-pwm）

PWM 扩展板（图 8-62）由 SparkFun 推出，它采用 TLC5940 IC，能够产生 16 个 PWM 输出。TLC5940 可以驱动 LED 或舵机发动机。它使用一个 SPI 类型的时控串行接口，但只用于接收数据。想获得更多信息与软件库，请前往 SparkFun 网站阅读相关资料。

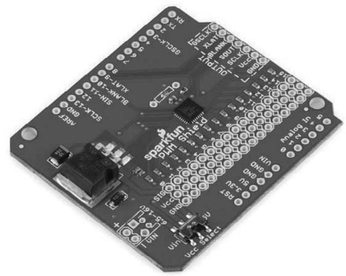


图 8-62：SparkFun PWM 扩展板

8.4.19 显示器

用于 Arduino 开发板的显示器扩展板包括 LED（发光二极管）读出器、LED 阵列、LCD（液晶显示器）、彩色图形显示器。一些扩展板使用了 Arduino 的多个数字输出，其他使用 SPI 或 TWI（I2C）接口。无论你想显示什么，都有可能找到相应的显示器扩展板帮助完成这项工作。

1. LED阵列

其实，单个 LED 阵列就已经很有意思了，如果再将其并排，则可以进一步形成色彩更为丰富的跑马灯效果。有关下面所列扩展板的更多信息，请参考相应网站。

- Adafruit LoL 扩展板（http://bit.ly/ada-lol）（图 8-63）

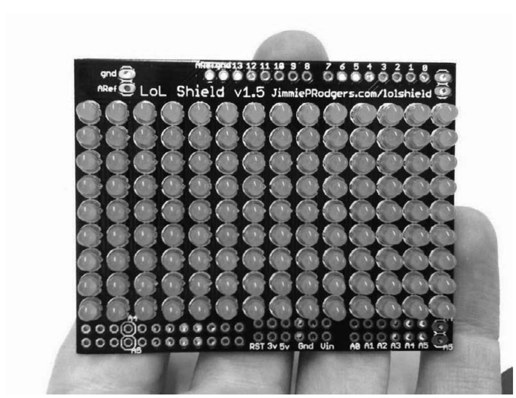


图 8-63：Adafruit 9 × 14 LED 阵列扩展板

- Solarbotics SMD LoL 扩展板（http://bit.ly/solarbio-smd-lol）（图 8-64）

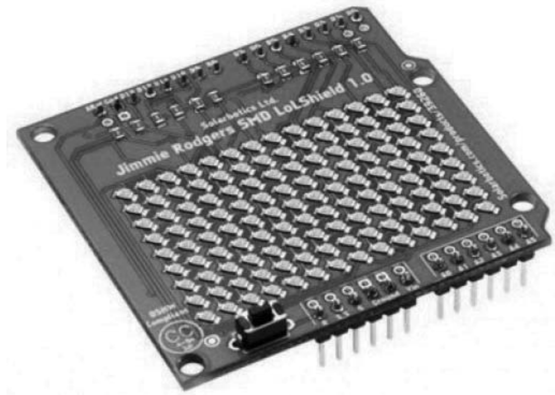


图 8-64：Solarbotics 9 × 14 LED 阵列扩展板

- Adafruit NeoPixel 扩展板（http://bit.ly/ada-neo）（图 8-65）

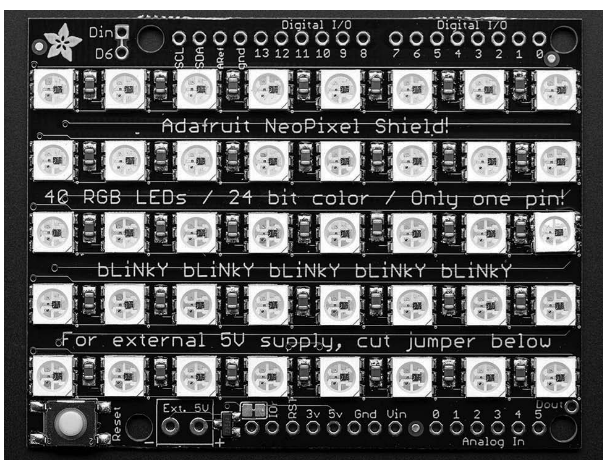


图 8-65：Adafruit 40 RGB LED 像素矩阵

2. 七段LED显示器

只要有 LED 的地方，几乎都能见到七段 LED 的身影。虽然现在看起来它有些过时，但当你需要又大又亮的数字显示，以便可以从房间的另一侧轻松看清时，七段 LED 显示器仍有用武之地。此外，在 8.4.22 节能了解到有关多功能扩展板的内容，其特色是一个 4 位数字 LED 显示器。

- 4x 7-Segment Arduino Compatible Digit Shield（http://bit.ly/arduino-7seg）（图 8-66）

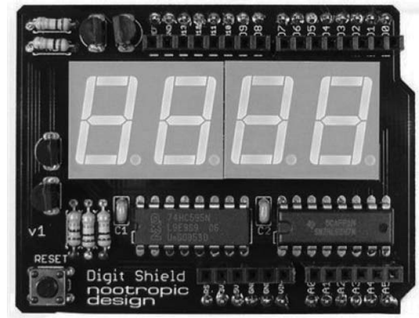


图 8-66：Nootropic Design 七段显示器扩展板

3. LCD显示器

许多低功耗的基于字符的 LCD 扩展板采用 16×2（每行 16 个字符，2 行）显示器，或白字蓝屏，或红字黑屏，或黑字绿屏。此外，还有其他组合可用，包括 16×4 与 20×4 配置。这些扩展板中的大多数都基于 Hitachi HD44780 LCD 控制器或者类似的控制器。

此外，还有像素可寻址显示器（pixel-addressable）与可显示位图（bitmap-capable）的 LCD 显示器可用，其中有些（比如 Nokia 5110）可以从多个供应商那里购买，它们往往都很容易连接到 Arduino。你也能比较轻松地找到分辨率为 128×64 与 160×128 像素的显示器，但能够用在 Arduino 兼容扩展板上的不是很多。关于显示器组件（非扩展板）的更多信息，请阅读第 9 章相关内容。

SainSmart LCD Keypad 扩展板（http://bit.ly/sainsmart-keypad）

这是一款常见的 LCD 扩展板（图 8-67），它采用 16×2 的 LCD 显示模块与 Hitachi HD44780 LCD 控制器（显示模块可以单独使用，第 11 章的信号发生器以及第 12 章的恒温器会用到）。



图 8-67：SainSmart 16 × 2 LCD keypad 扩展板

该 LCD 扩展板为 5 个按钮开关使用一个电压分压器，每个按钮按下都会产生不同电压。图 8-68 显示了它是如何工作的。这个方案的好处是，5 个开关输入都通过一个模拟输入传送。

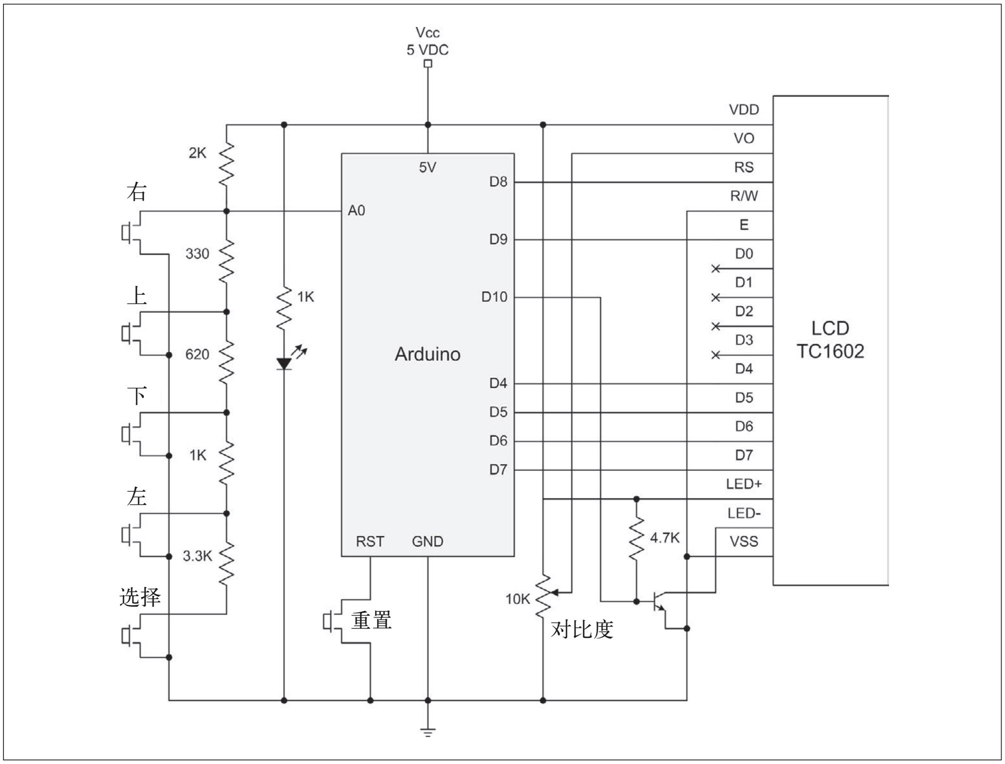


图 8-68：SainSmart LCD 扩展板电路图

DFRobot LCD Keypad 扩展板（http://bit.ly/dfrobot-keypad）

该扩展板类似于 SainSmart LCD 扩展板，但它带有到底部 Arduino 的模拟引脚连接（见图 8-69）。根据供应商提供的技术文档，额外的引脚主要用来连接 APC220 无线模块或蓝牙模块。



图 8-69：带有模拟引脚的 DFRobot 16 × 2 LCD keypad 扩展板

Adafruit LCD 扩展板套装（http://bit.ly/adafruit-lcd）

这款 16×2 LCD 扩展板套装（图 8-70）出自 Adafruit，它采用 I2C 接口与 MCP232017 I/O expander IC（第 10 章也会用到）控制 LCD 显示器。这导致它只能使用 Arduino 的 2 个引脚 A4 与 A5 作为 I2C 接口。LCD 与按钮全部连接到 MCP23017 IC 上，它并不使用电阻分压器。请注意，这是一个套装，但组装并不太难。

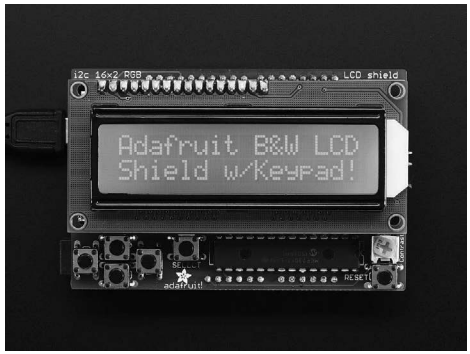


图 8-70：带有 16 × 2 字符显示的 Adafruit LCD 扩展板套装

Nokia LCD5110 Module with SD（http://bit.ly/nokia_lcd）

该扩展板（图 8-71）集成了一个 Nokia 5110 LCD 显示器，带有一个 SD 闪存卡插口。它将引脚 D3、D4、D5、D5、D7 用作 LCD 显示，将 D10、D11、D12、D13 用作 SD 卡。引脚 D0、D1、D2 可用作其他用途。

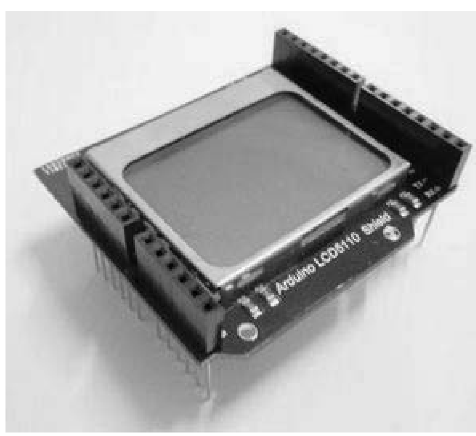


图 8-71：Elechouse 的 Nokia 5110 LCD 扩展板

与 16×2 的字符显示器不同，5110 是一个可以显示图形的 LCD，显示区域为 48×84。它原本用在手机上，目前所有可用的都是从原来的库存中剩下的，所以有些有划痕，有些可能还有一些小瑕疵。如果它们出了什么毛病，请不要感到意外。因此，为了使用它们而设

计一款新产品并不是一个好主意，但它们的确很“好玩”，并且价格相对低廉。

4. TFT显示器

TFT LCD（薄膜晶体管液晶显示器，简称为 TFT）是一种常见的显示器，广泛应用于计算机显示器、收银机显示器、手机、平板电脑，以及其他需要显示彩色图形的地方。用于 Arduino 的彩色 TFT 扩展板能够以 240 × 320 像素的分辨率显示数千种颜色。此外，还有更大的显示器可用，但它们大都不适合放在扩展板上。

TFT 扩展板一般比较便宜，大部分使用 SPI 接口，并且一些为生成高速图像提供了并行数字接口。

ITEAD 2.4″ TFT LCD Touch 扩展板（http://bit.ly/ITEAD-tft）

该扩展板（图 8-72）采用并行数字接口与 Arduino 进行通信。它有一个 S6D1121 TFT 控制器，支持 8 位接口，触屏功能由 TSC2046 芯片进行处理。更多细节请访问供应商网站。



图 8-72：带有触摸屏的 ITEAD 2.4 inch color TFT 扩展板

Adafruit 2.8″ TFT Touch 扩展板（http://bit.ly/ada-tft）

这块 2.8 in 扩展板（图 8-73）由 Adafruit 推出，它为 ILI9341 显示器控制器（带有内置视频 RAM 缓冲）与 STMPE610 触屏控制器提供高速 SPI 接口。同时，它也集成了一个 microSD 闪存卡槽。该扩展板使用 Arduino 的数字引脚 D8~D13，触屏控制器使用引脚 D8，microSD 卡槽选择在引脚 D4 上。



图 8-73：带有电阻式触摸屏的 Adafruit 2.8 inch TFT 扩展板

8.4.20 仪表扩展板

尽管在数量上不及其他类型的扩展板，但仍然有一些仪表类型的扩展板可用，其中包括数据记录扩展板、逻辑分析器、高精度模数（A/D）转换器。此处所说的“仪表”可以从周围物理世界感知与捕获数据，或者产生模拟信号。

带有板载数据捕获与转换能力的扩展板并不太多，主要因为 Arduino 上的 AVR MCU 本身已经拥有其中大部分功能。Arduino 开发板上的 AVR MCU 内置有 A/D 转换器（ADC），它是 10 位 ADC，即分辨率为 1/1024（每个 DN）。如果想要更高的分辨率（12、16 或者 24 位），可以考虑使用一些外加模块或扩展板。

Adafruit Data Logging 扩展板（http://bit.ly/1Tjlu5V）

Adafruit 数据记录扩展板包括一个 SD 闪存卡槽与实时时钟（RTC）芯片。当来自 Arduino 的主电源关闭时，RTC 可以使用电池供电。此外，它还提供了一小块原型区域，供用户搭建自己的电路。该扩展板不支持堆叠功能。

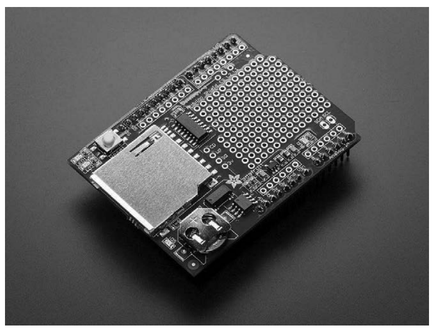


图 8-74：Adafruit assembled 数据记录扩展板

Adafruit Ultimate GPS Logger 扩展板（http://bit.ly/ada-gps）

这款扩展板（图 8-75）搭载了一个 GPS 接收器，还有一个 SD 闪存卡槽、一个 RTC 芯片。GPS 输出的数据能够被自动记录到闪存卡。该扩展板不支持堆叠功能。

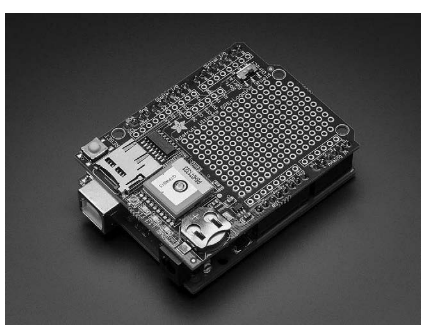


图 8-75：Adafruit GPS 数据记录扩展板

HobbyLab Logic Analyzer and Signal Generator 扩展板（http://www.arduinolab.us）

事实上，它是扩展板上一个独立的逻辑分析器。可以监视 Arduino 信号，但不会干扰它们，这让我们很容易了解 I/O 引脚上发生了什么。除了逻辑分析器功能之外，它也包含一个 SPI 解码器、一个 UART 解码器、一个单线监视器。它并不直接与 Arduino 通信，而使用 USB 接口与主计算机进行交互。该扩展板支持堆叠功能（见图 8-76）。

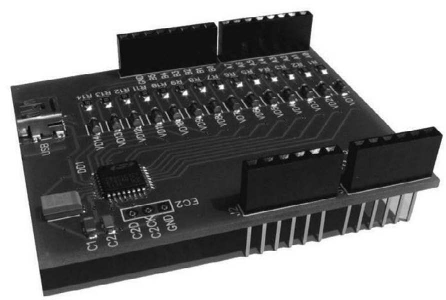


图 8-76：HobbyLab 逻辑分析器与信号发生器

Iowa Scaled Engineering 16-Channel 24-Bit ADC 扩展板（http://bit.ly/ard-ltc2499）

该扩展板集成了一个 24 位 A/D 转换器与一个精密电压参考源，以便从多个单端或差分输入获得读数。它也搭载有板载 EEPROM，用于存储、读取校准与配置数据，通过 I2C 接口（TWI）与 Arduino 进行通信。该扩展板支持堆叠功能（见图 8-77）。

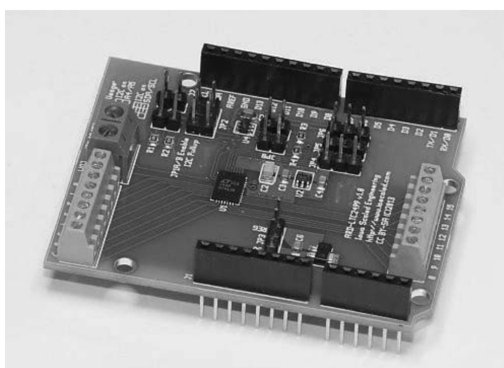


图 8-77：Iowa Scaled Engineering 24-bit ADC 数据采集扩展板

Visgence Power DAC 扩展板（http://bit.ly/power-dac）

AVR MCU（包括搭载着 avr mcu 的 arduino）所缺的一个功能是内置数模转换器。AVR 内部 ADC 集成了一个 10 位的 DAC，但其输出无法在外部使用。尽管音频输出扩展板容易购得，但看上去好像没有太多纯 DC 输出 DAC 扩展板可用。Visgence Power DAC 扩展板（图 8-78）提供 3 个通道的模拟输出，允许高达 250 mA 的电流通过。

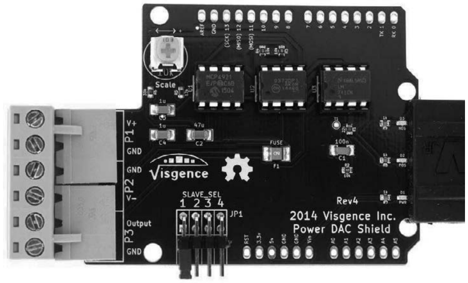


图 8-78：Visgence 12-bit Power DAC 扩展板

8.4.21 适配器扩展板

适配器扩展板在两个物理不兼容的模块之间充当物理接口。适配器扩展板与信号路由扩展板（8.4.5 节）最主要的不同在于，就本章中关于如何组织扩展板而言，适配器是作为物理接口使用的，而信号路由扩展板不会解决物理差异，它只处理信号。

Tronixlabs Australia Expansion Shield for Arduino Nano（http://bit.ly/exp-nano）

Nano 与标准大小的 Arudino 功能完全一样，但不能与传统扩展板一起工作。为了解决这一问题，该扩展板（见图 8-79）将引脚从 Nano 引出到排针或者标准的引脚插口连接器。

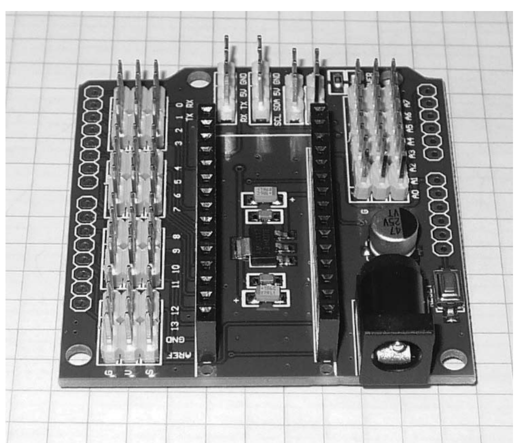


图 8-79：Tronixlabs Nano 适配器扩展板

Arduino Nano I/O Expansion Board（eBay）（http://bit.ly/exp-nano-ebay）

这是另一种 Nano 适配器（图 8-80），你可以在 eBay 上找到它，此外还有其他类似的可以使用。

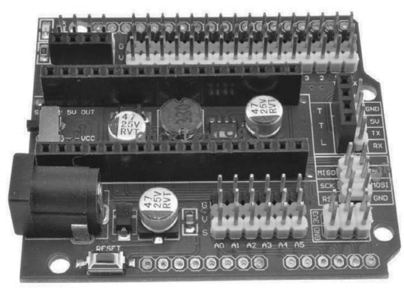


图 8-80：Arduino Nano 扩展板

Screw Terminal 扩展板（http://bit.ly/screw-term）

尽管从技术上看这不是一个扩展板，但这些容易获取的端子适配器本身允许你将多达 18 标准尺的绝缘导线连接到 Arduino。你可以很容易地从多种渠道购买。另外，请注意这些器件具备堆叠能力（见图 8-81）。

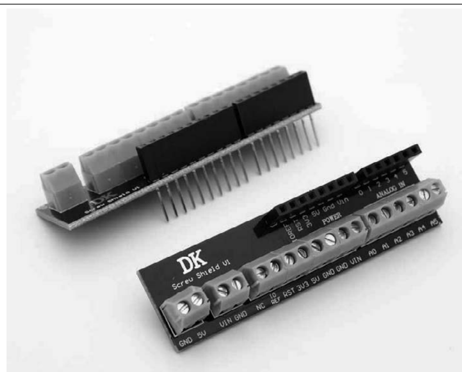


图 8-81：端子适配器

8.4.22 混杂扩展板

本部分将介绍一些有用的扩展板，它们不属于前面任何一个分类。我喜欢称之为“翼板”（wing shields），因为它们允许更整齐的布线，这里出现的多功能扩展板有许多用途。

Adafruit Proto-ScrewShield（Wingshield）（http://bit.ly/ada-proto-screw）

该 Wingshield（亦称为 Proto-ScrewShield）是一个无源扩展板，带有两套小型端子台（见图 8-82）。如果你打算将 Arduino 集成到一款商业产品或实验环境中，那么可能需要使用这样一个扩展板。与引脚跳线相比，端子台中的螺钉可以提供更安全、可靠的连接。该扩展板中间有一块原型区域，可以用于安装各种传感器模块，也可以用于搭建自己的电路。该扩展板包含一个重置按钮、一个 LED，并且支持堆叠功能。请注意，它是一个套件，由裸板 PCB 与一包元件组成。组装并不难，但需要有一点焊接技术。

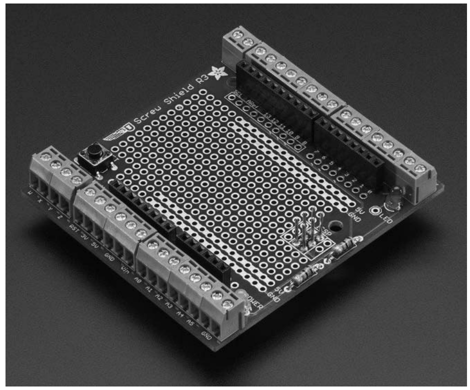


图 8-82：Adafruit Wingshield

DFRobot Screw ProtoShield（http://bit.ly/screw-protoshield）

类似于 Adafruit Wingshield，该扩展板（图 8-83）为每个来自 Arduino 的信号提供螺丝式端子台，但它已经是组装好的成品。此外，它支持堆叠功能，你可以将其放到其他扩展板之下，并且其端子台仍然可以正常访问。

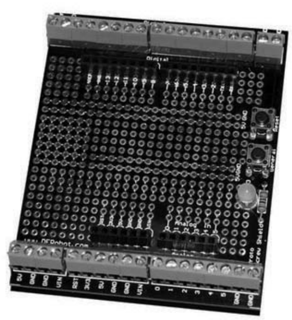


图 8-83：DFRobot 端子台扩展板

DealeXtreme DIY Multifunction 扩展板（http://bit.ly/diy-multi）

这款扩展板很有趣：它有一个 4 位数字 LED 显示器、一个用于 APC220 蓝牙模块的接口点、几个连接到 Arduino 引脚 A1~A3 的开关按钮、一个重置按钮，以及一个连接到 A0 输入的电位器（见图 8-84）。4 个 LED 分别连接到数字引脚 D10、D11、D12、D13，一个 3×4 排针将引脚 D5、D6、D9、A5、+5 V 与地引出。令人遗憾的是，这款扩展板的文档资料相当少，你要花费很大力气才能获取有用信息。在 HobbyComponents.com（http://bit.ly/dx-diy-hc）论坛可以找到更多内容。从 http://bit.ly/dx-diy-sketch 可以下载一些示例代码与电路图。ZIP 文档中的目录名称都是中文的，其中大部分程序都带有英文注释。

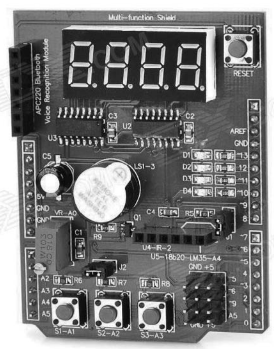


图 8-84：DX DIY 多功能扩展板

关于多功能扩展板各种组件的布局情况，请参考图 8-85。请注意，该扩展板并不支持堆叠功能，这也在情理之中，因为在其上叠加另一个扩展板将使 LED 显示器变得毫无用处。

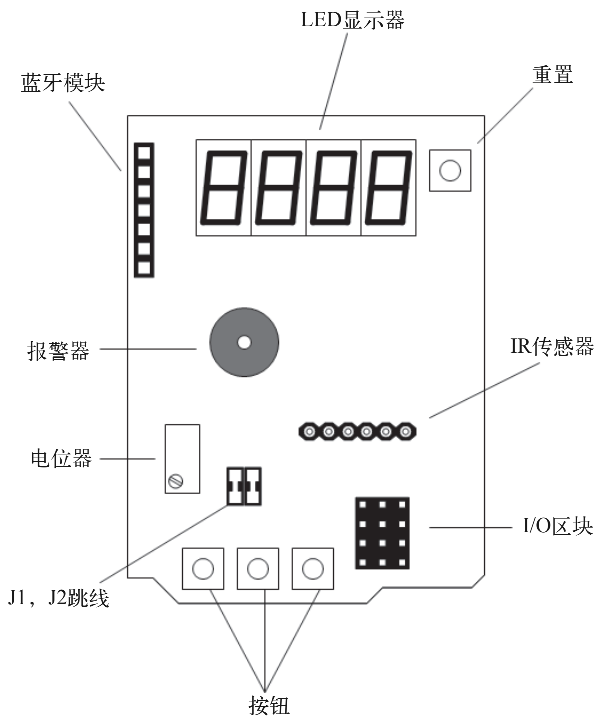


图 8-85：多功能扩展板主要功能特点

该扩展板提供的电路图有点模糊不清，所以我创建了一个替换版本，如图 8-86 所示。请注意，该扩展板会占用 Arduino 的所有引脚。表 8-1 列出了 Arduino 引脚以及多功能扩展板用它们来做什么。

这个扩展板是一个好例子，当你遇到一个新扩展板时，常常遭遇类似情形：相关文档可能很少，并且已有文档也大都使用你不懂的语言编写（比如英文或者其他语言）。该扩展板的电路图正确，但粗看之下很难理解，并且没有有关引脚的细节描述（其实现在有了）。

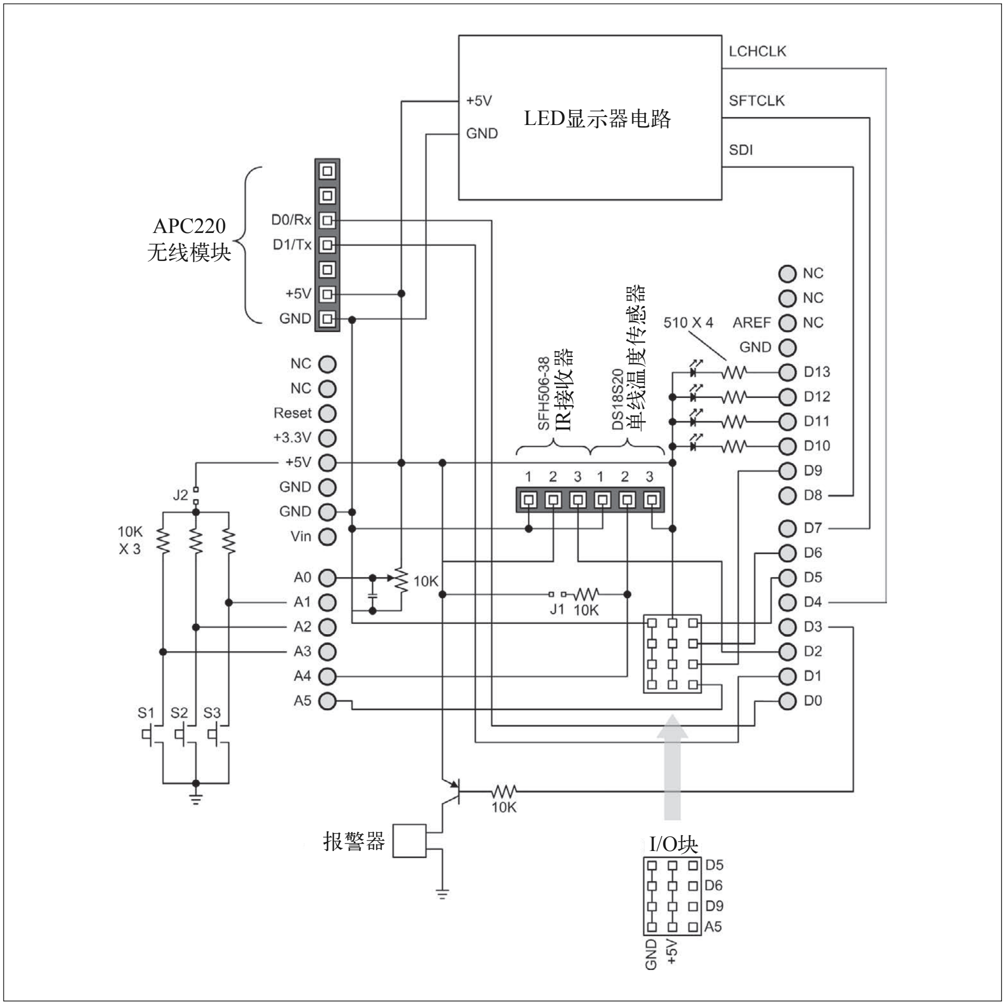


图 8-86：多功能的扩展板电路图

表8-1：多功能扩展板Arduino引脚功能

引脚	功能	引脚	功能
D0	来自无线模块的 Rx	D5	到 I/O 区域的 D5
D1	到无线模块的 Tx	D6	到 I/O 区域的 D6
D2	ID 接收输入	D7	LED 显示器时钟
D3	报警器控制	D8	LED 显示器串行数据输入
D4	LED 显示锁存	D9	到 I/O 区域的 D9

（续）

引脚	功能	引脚	功能
D10	LED	A1	开关 S3
D11	LED	A2	开关 S2
D12	LED	A3	开关 S1
D13	LED	A4	温度传感器输入
A0	电位器游标	A5	I/O 区域中的 A5

8.5 非常见Arduino扩展板

除了大量应用于常见场合（从 RS-232 I/O 到 PWM 舵机控制）的扩展板之外，还有许多特殊用途的扩展板。一些开源 3D 打印机采用 Arduino 作为主控制器，控制接口（某种形式的扩展板）也能很容易地找到。这些扩展板中的一些是针对 Arudino Mega 系列开发板的，并对大小做了相应调整。

还有其他一些非常见的扩展板，它们不符合本章中的任何一种类型，但非常有趣，可玩性很高。Gameduino 就是其中之一，它为 FPGA（现场可编程门阵列）芯片提供了一个“载体”，并且拥有其他应用潜力，比如视频游戏。

Qunqi CNC Shield for Arduino V3 Engraver（http://bit.ly/qunqi-cnc）

该扩展板（图 8-87）不带电机驱动模块，但你可以很容易地购买。标准的驱动器模块采用 Allegro 的 A4988 DMOS 微步电动机驱动芯片。

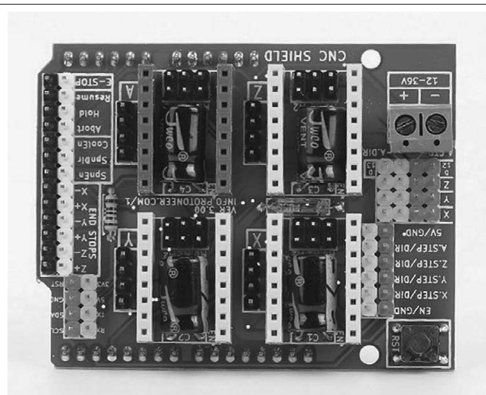


图 8-87：Qunqi A4988 driver CNC 扩展板

SainSmart RepRap Arduino Mega Pololu 扩展板（http://bit.ly/sainsmart-reprap）

RepRap 3D（http://reprap.org）是人类历史上第一部通用的自我复制型机器，是一台小型 3D 打印机，可以为其他 RepRap 机器打印零件，也可以打印其他许多有用的东西。该扩展板（图 8-88）被设计用于取代 RepRap 设备上的电子器件，它使用 Arduino Mega 作为处理器。

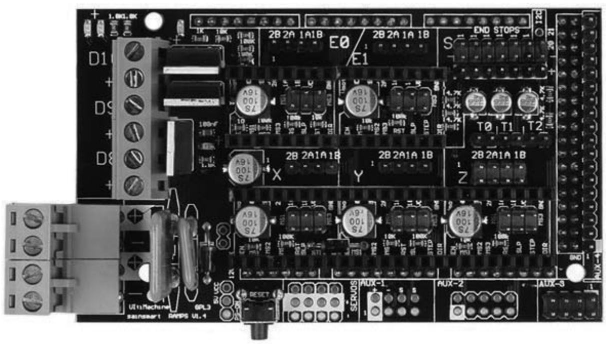


图 8-88：SainSmart RAMPS 1.4 RepRap 扩展板 -3D 打印机

excamera Gameduino（http://bit.ly/excam-gameduino）

该扩展板（图 8-89）使用 Xilinx FPGA 为自制游戏控制台控制图形与声音。Arduino 用于连接控制部件，管理游戏运行。Gameduino 是开源的，所有技术细节均已公开。你可以根据需要对 FPGA 稍加修改，以将其应用于游戏以外的领域。

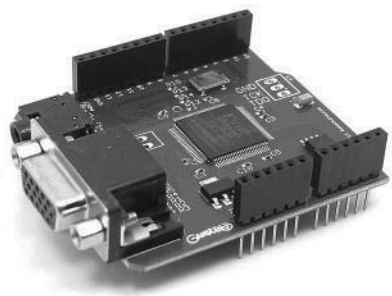


图 8-89：excamera Gameduino 游戏控制器扩展板

8.6 资源

表 8-2 列出了本章涉及的供应商与生产商。当然，还有许多其他厂商，从他们那里也可以找到有用的或者新的扩展板。在谷歌搜索栏中输入 Arduino shield，将会返回大约 400 000 个结果，所以你可以在很多地方找到需要的产品与信息。不要因为此处未列出供应商或生产商而不考虑他们。只是，若要将这个瞬息万变的市场上的全部产品囊括其中，恐怕本书将永无尽头。

表8-2：扩展板供应商与厂商列表

名称	URL	名称	URL
Adafruit	www.adafruit.com	Macetech	www.macetech.com/store/
Arduino	store.arduino.cc	Mayhew Labs	www.mayhewlabs.com
Arduino Lab	www.arduinolab.us	Nootropic Design	www.nootropicdesign.com
Circuits@Home	www.circuitsathome.com	Numato	www.numato.com
CuteDigi	store.cutedigi.com	RobotShop	www.robotshop.com
DFRobot	www.dfrobot.com	Rugged Circuits	www.ruggedcircuits.com
DealeXtreme (DX)	www.dx.com	SainSmart	www.sainsmart.com
Elecfreaks	www.elecfreaks.com	Seeed Studio	www.seeedstudio.com
Elechouse	www.elechouse.com	SparkFun	www.sparkfun.com
excamera	www.excamera.com	Tindie	www.tindie.com
Iowa Scaled Engineerin	www.iascaled.com	Tronixlabs	www.tronixlabs.com
iMall	imall.itead.cc	Vetco	www.vetco.net

第 9 章

模块与I/O组件

虽然许多 Arduino 扩展板已经内置了大量有趣且有用的功能，但它们并非无所不能，也不应该无所不能。因为有大量不同类型的传感器、控制器、驱动器接口可用，这些都能与 Arduino 配合使用。许多供应商为 Arduino 提供单一功能的附加传感器组件和小型 PCB 模块，其中包括温度与湿度传感器、震动探测器、光电探测器、键盘、摇杆，甚至固体激光器。

几乎所有能与微控制器一起使用的传感器、控制器、驱动器设备都能与 Arduino 一起使用，只是对 DC 供电电压有一些限制，具体取决于 Arduino 中使用的微控制器类型（3.3 V 与 5 V）。但是，对于大部分部件，这是相对次要的细节，只要使用简单的接口电子器件与适当的电源供给就能解决这些问题。

本章将学习各种 I/O 模块与独立部件。I/O 模块是小型 PCB，用于执行特定功能，它们只使用几个有源组件——如果要用。I/O 模块一般较小，大约只有邮票大小或者更小，使用引脚进行连接。它们可以与“母 – 公”（female-to-male）或“母 – 母”（female-to-female）跳线工作得很好，有时可以使用专门的多线电缆将模块连接到符合特定用途的扩展板上。此处会涉及来自 KEYES、SainSmart、TinkerKit 的产品，主要因为它们都是常用且比较有代表性的模块。其他值得考虑的模块还有 Grove 系列模块、Seeed Studio 的接口板，以及 TinyCircuits 推出的各种模块。

独立 I/O 组件覆盖领域相当广泛，从 LED 到图形显示器，从机械传感器（比如开关与干簧继电器）到独立的温度与湿度传感器。讨论完模块之后，我们还将特意讲解一些独立传感器，因为许多模块都会使用这些组件。不同部分之间会适当地提供交叉引用，以帮助各位更好地理解所讲内容。

大多数人都崇尚简洁与可靠。然而令人遗憾的是，使用随处可见的跳线连接模块与其他组件一点也不整齐，不简洁，并且位于每条跳线末端的按钮式压接连接器都有从模块引脚松开的倾向。

请不要直接焊接模块引脚，也不要使用硅粘合剂固定跨接连接器（jumper connector）。在此情形下，可以使用模块式连接器（modular connector）。借助简单的手工工具，你可以为自己的特定应用提供定制，或者也可以选用 TinkerKit、Grove、TinyCircuits 等系统。本章结尾部分会简单介绍如何将模块连接到 Arduino，当然，是不使用跳线的方法。

9.1 模块

要想将传感器、开关、继电器、话筒连接到 Arduino 并进行试验，最简便的方法无疑是使用传感器与 I/O 模块。图 9-1 显示了一些不同类型的模块。

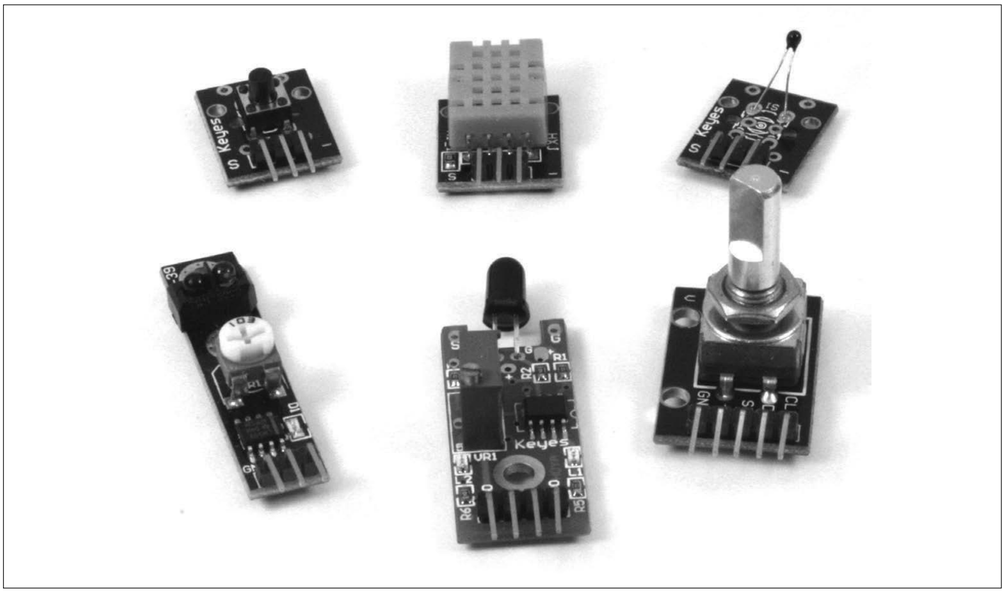


图 9-1：不同类型和大小的传感器模块

随着时间的推移，最终你会得到一大堆模块，其中有些要比其他一些有用得多。我建议你购买能负担得起的最大模块套装，然后从中找出自己最常用的，把它们放在手边，并将其他一些较少用到的模块保存起来，以便在将来的项目中使用。

> 请注意，网上那些介绍各种模块工作原理的文档并非总是正确的，其中可能包含翻译错误。比如，有时在线文档会说某个模块工作时会产生高电平输出，而实际上它产生的是低电平输出。请记住，将一个模块加入到某个电路之前，一定要先用数字万用表（DMM）检测它能否正常工作。本部分提到的大部分模块都经过了测试，以确定它们是否都能正常工作，而且此处的说明都能反映这些测试结果。尽管如此，对于那些与本章所列模块类似的模块，我也不能说它们都有完全相同的行为或一样的引脚功能。在这一领域，标准化仍需抓紧。

至于这里所列模块的电路图，我真的没有找到任何官方图示。网上的几个“大牛”主动承担了探寻一些模块 PCB 踪迹的任务。我也尝试着收集自己能找到的一切资料，并努力通过逆向工程技术把它们拼合在一起。有时会得到一个完整的电路图，更多时候，我只是想验证引脚是否真的能像文档描述的那样工作。

9.1.1　物理外形

各种模块的大小各不相同，从 1.8 cm × 1.5 cm（大约 3/4 in × 9/16 in）到 4.2 cm × 1.7 cm（大约 $1\,^{11}/_{16}$ in × 5/8 in）不等。有些高达 3.4 cm × 2.5 cm（大约 $1\,^{5}/_{16}$ in × 1 in），比如 5 V 的继电器模块。图 9-2 显示了各种模块的大小。请注意，这些大小都是近似值，实际的 PCB 可能会有大约 +/−1 mm（0.04 in）的误差，具体取决于模块生产地。

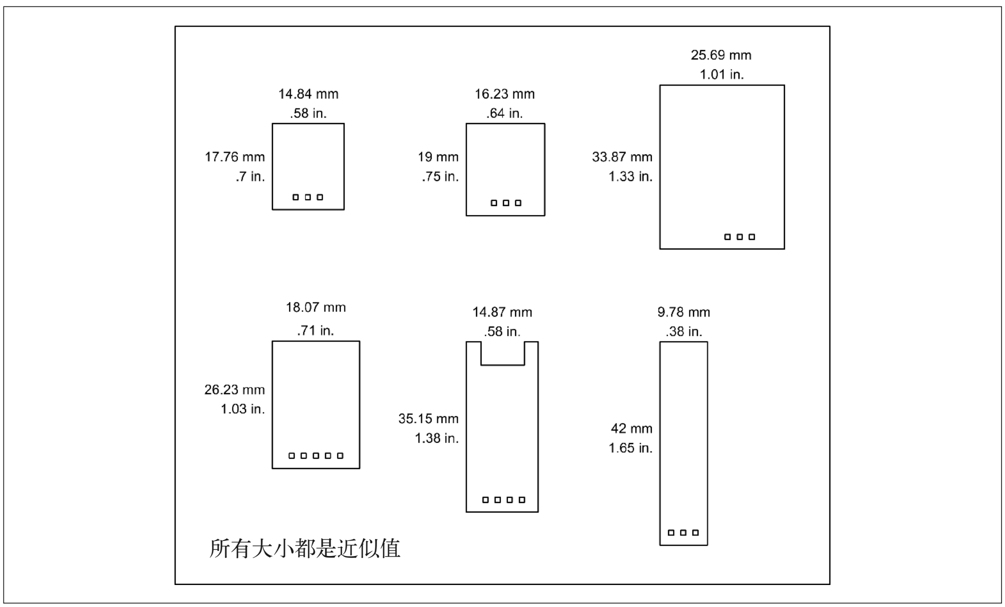


图 9-2：一些常见的模块大小

许多模块 PCB 上都带有安装孔洞，方便使用 #2 机械螺丝进行固定。一般使用米制等量表示为 M1.6 或 M1.8。如图 9-3 所示，两个模块通过 2-56 机械螺丝与尼龙绝缘管固定在一起。单线温度传感器与水银倾斜开关模块恰好就采用这种堆叠固定方式。

令人遗憾的是，并非所有模块都能很好地支持堆叠功能。有时，一些安装孔的间距不一样，或者位于 PCB 的错误位置，以致于无法让模块实现堆叠。有些模块干脆不带安装孔，所以堆叠模块前一定要检查。

9.1.2　接口

各种 PCB 模块使用的引脚各不相同，具体取决于模块类型。除了 TinkerKit 系列之外，确实还有许多没有实现标准化。图 9-4 列出了你可能遇到的一些引脚配置情况。

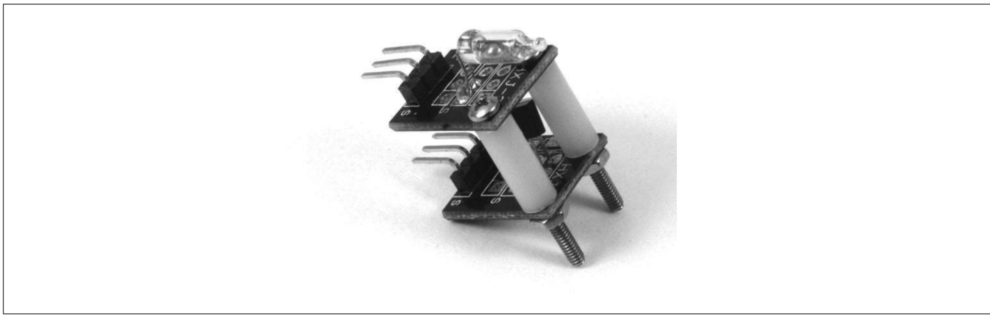


图 9-3：使用 #2 机械螺丝与绝缘管固定模块

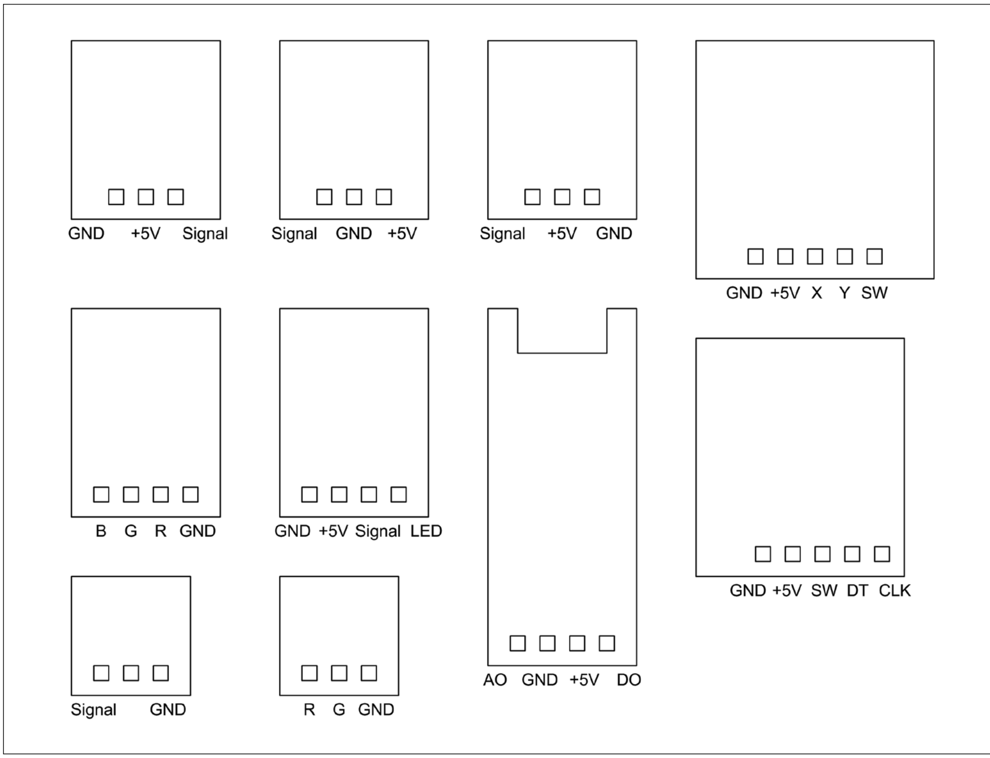


图 9-4：模块中常见的引脚配置

虽然你可能认为带有 3 个或 4 个引脚的模块会与带有模块连接器或 I/O 引脚区（比如第 8 章介绍的那些）的 I/O 扩展板相兼容，但情况并非总是如此。在模块与扩展板之间，当连接一系列模块到接口扩展板（被设计为与这些组件一起工作）时，“引脚 – 引脚”（pin-to-pin）的兼容是唯一可以确保的。TinkerKit 就是其中一例，但仅局限于将 TinkerKit 模块连接到 TinkerKit 接口扩展板。连接模块之前，请一定要检查模块引脚，确保电压与信号都正常。

由于 AVR 是一个相当强大的设备，所以你可以将许多传感器直接连接到 Arduino 的输入引脚，当然也可以连接一些输出设备。固体激光器或 RGB LED 这类耗电型输出设备使用的电流超过了 Arduino 的 AVR 芯片所能直接提供的电流（请回顾第 3 章中有关电源和电流规格的内容），为此，需要使用某种驱动器以提高电流。

有些输出模块内置有大电流输出接口，有些则没有。如图 9-5 所示，一个简单的电路能安全地提高电流，以便提供给继电器或激光 LED 使用。图 9-5 右侧的电路中，R 值将由 LED 及其工作所需的电流大小决定。只要电流未超过晶体管的额定值，整个电路都会正常工作。

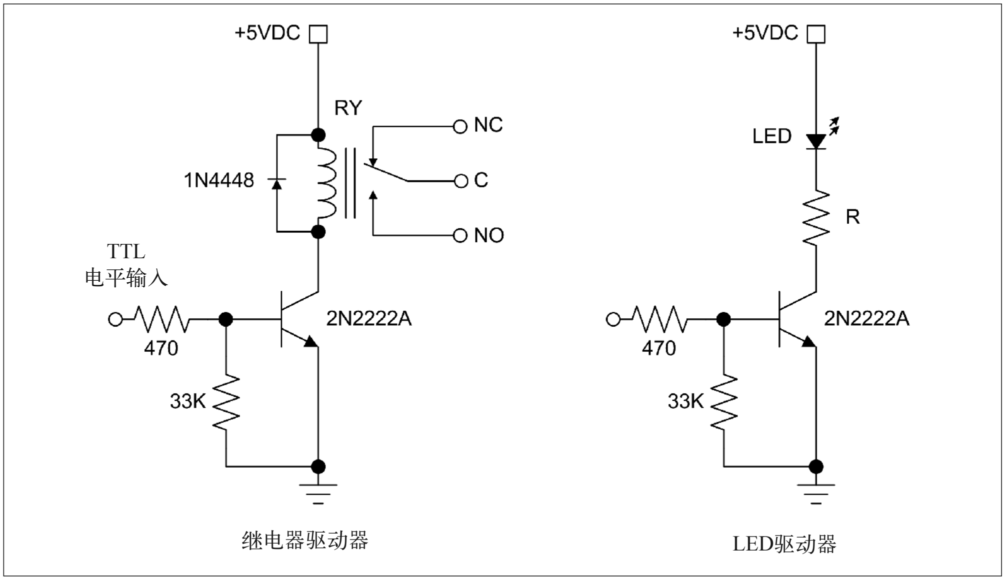


图 9-5：驱动大电流设备的输出电路

另一个选择是使用专用 IC，比如 MAX4896，如图 9-6 所示。它使用一个 SPI 接口，Arduino 能直接连接到该 IC 上。虽然该 IC 被设计用于驱动小型继电器，但它完全可以驱动大型 LED。

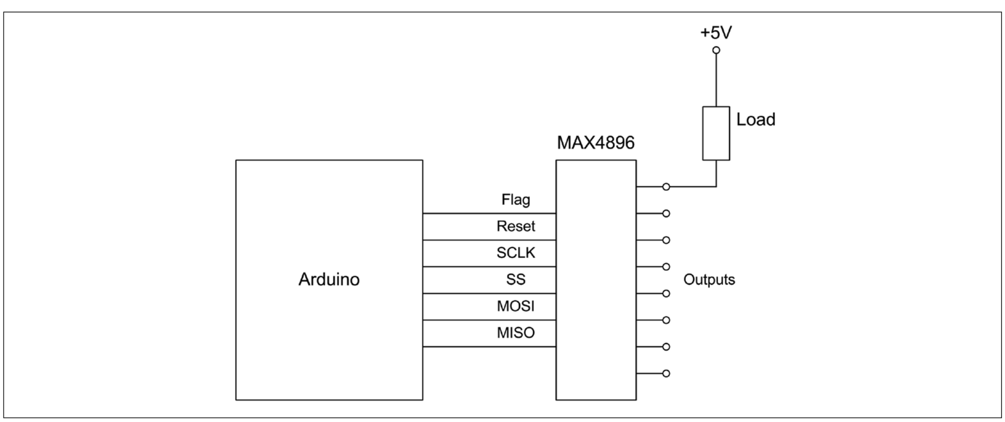


图 9-6：多个继电器或其他负载的输出驱动器 IC

9.1.3　模块来源

你可以从表 9-1 列出的供应商与厂商那里购买单个 PCB 模块与带有 24、36 或更多模块的传感器套装，也可以在 eBay 与 Amazon 上找到它们。图 9-7 显示了一个塑料盒，里面装着一整套传感器模块（总共有 37 种，最后一个格子里放着 2 种模块）。



图 9-7：一整套输入 / 输出模块

表 9-1 列出的大部分供应商也销售单个模块，包括跳线、互联电缆以及“裸”的输入与输出组件。尽管不是套装中的每个模块都可以单卖，但你仍然能以少量购买方式买到自己所需的大部分模块。

表9-1：部分传感器与输出模块供应商与厂商

名称	URL
Adafruit	www.adafruit.com
CuteDigi	store.cutedigi.com
DealeXtreme (DX)	www.dx.com
KEYES	en.keyes-robot.com
SainSmart	www.sainsmart.com
Seeed Studio	www.seeedstudio.com
TinyCircuits	www.tiny-circuits.com
Trossen Robotics	www.trossenrobotics.com
Vetco	www.vetco.net

9.1.4　模块说明

本部分涉及的模块有 3 个来源，分别为 KEYES、SainSmart、TinkerKit。由于篇幅限制，这里的讲解都比较简单，主要强调表中列出的各种模块的物理外形与电气连接。9.4 节将

讲解有关传感器模块的更多细节，适当的时候会添加对模块的交叉引用，以提供更多细节描述。

使用模块一段时间之后，你可能会注意到此处介绍的许多模块都使用了相同的基本电路，一般由一个 LM393 比较器与某种传感器组成。在每种情况下，电位器设置比较电压阈值，LM393 的输出被连接到模块的信号引脚。在表 9-2 中，你可以看到带有类似电路的 KEYES 模块；而在表 9-3 中，你能看到带有类似电路的 SainSmart 模块。

表9-2：带有类似电路的KEYES模块（图9-8）

产品编号	名称
KY-025	簧片开关模块
KY-026	火焰传感器
KY-036	导电接触传感器
KY-037	灵敏话筒传感器
KY-038	话筒传感器

表9-3：带有类似模块的SainSmart模块（图9-8）

产品编号	名称
20-011-981	光敏传感器
20-011-982	震动传感器
20-011-983	霍尔效应传感器
20-011-984	火焰传感器

图 9-8 给出了 LM393 比较器电路的通用电路图，表 9-2 与表 9-3 列出的模块就使用这种比较器电路。实际的元件值可能略有不同，但多种模块都使用这种基本电路。多个模块之间最主要的不同是传感器（IR 火焰、话筒、LDR 等）。

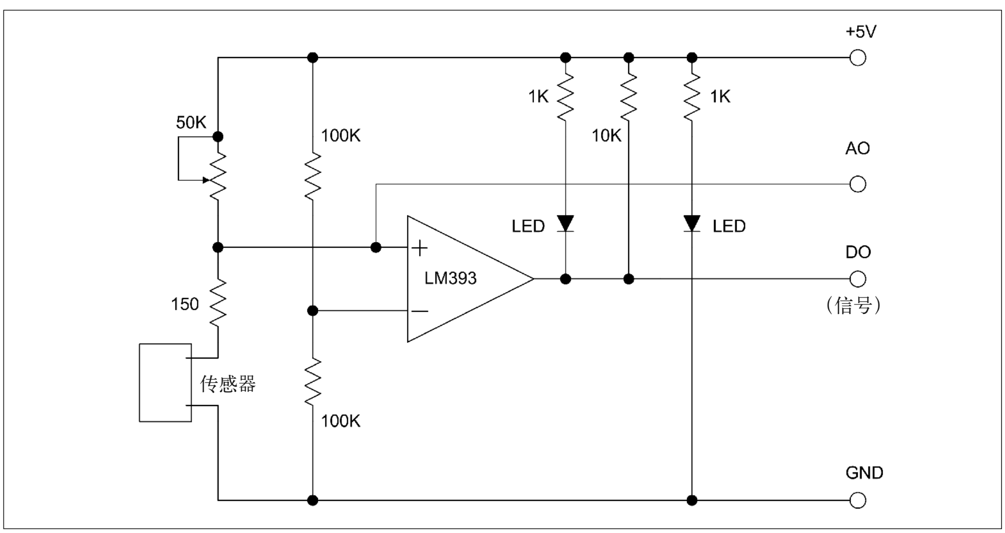


图 9-8：带有比较器 IC 的通用模块电路

DO（数字输出）端子直接来自于比较器。当同相输入（+）大于反相输入（–）时，比较器输出高电压；而同相输入（+）小于反相输入（–）时，比较器将输出低电压。一些模块被设计为使用比较器 IC 的输入，它被安排在图 9-8 的对面，但工作原理是一样的。

同相输入的电压由电位器或可调电阻进行设置。它是分压器的一半，传感器与一个限流电阻组成另一半。另一个分压器将 +5 V VCC 的一半应用到反相输入，大约是 2.5 V，用作参考输入。可调电阻用于设置电压，在该电压点，比较器将随着传感器修改其阻值而改变自身的输出，以对某种输入做出响应。

在电路的输出端，10K 电阻用作 IC 输出引脚上的上拉电阻。AO（模拟输出）是原始值（raw value），它来自 LM393 IC 输入端的传感器（本示例中）。在这些只有单输出的模块中，只有比较器的输出被引出到终端引脚，产生 DO 信号。一个模块也可以带有一个 LED，用于指示电源接通状态；还有一些有额外的 LED，用于指示比较器输出为低电压（它成为 LED 的电流）。

所有这些如何与传感器相关，将取决于传感器如何对输入做出响应。你可能需要对此做些实验，以便了解它是如何工作的。在许多情况下，有源传感器会表现出电阻下降，这会引起同相输入低于反相输入的参考电压。发生这种情形时，比较器电路（图 9-8）的输出就会降低，IC 输出上的 LED 将会激活亮起（IC 用作 LED 的电流）。

当输入处于活动状态或低于某个阈值时，比较器电路就表现为低电平输出，称为“低电平有效电路”（active-low circuits）。在这样的电路中，输出脚上的低电压等价于真值条件。当传感器处于运行状态，或者输入高于某个阈值时，比较器电路表现为高电平输出，称为“高电平有效电路”（activehigh circuits）。在高电平有效电路中，高电平输出等价于真值条件。从这个意义上讲，真或假（true 或 false）只代表传感器是否接收输入，“真”表示接收，“假”表示不接收。

> 将一个模块连接到 Arduino 之前，要花一些时间使用 DMM 与放大镜仔细检查模块。如果购买的模块非常便宜，认真检查就显得特别重要。在这样的模块上，我曾经发现部件缺失、焊点间存在短路，以及连接引脚无法使用等问题。我甚至在检查过的一个模块上发现，有一段导线将连接 +5 V 的迹线与接地引脚焊接在一起。这些都会导致一些问题。一个模块拥有接地引脚、+5 V、信号引脚并不表示所有引脚都会被用到，也不意味着所有引脚都会执行其名称所标注的功能。处理完这些问题之后，模块往往可以工作得很好（毕竟它们只有少量零件）。记得检查并做记录，这样可以帮你日后远离可能出现的烦恼！

中国深圳的 KEYES DIY Robot Co. Ltd（简称为 KEYES）公司设计的许多常用模块都可以单独购买，当然，你也可以直接购买它们的模块套装，每个套装一般含有 36 个模块。你可能也会碰到带有 HXJ 字样的模块，它们与 KEYES 模块拥有的功能完全一样，但 PCB 布局可能略有不同。

请看表 9-4，它列出了本部分要讲解的 KEYES 模块。表 9-7 给出了 KY-001~KY-040 模块的图片、说明及引脚图，其中显示的一些型号来自 KEYES，一些来自其他供应商（比如

HXJ 器件），但它们其实完全一样，都拥有相同的引脚与功能，往往有一样的名称（KY-002、KY-027 等）。请注意，并不存在 KY-007、KY-014、KY-029、KY-030 模块，我也不知道其中原因。

表9-4：常用的KEYES I/O模块

产品编号	名称	产品编号	名称
KY-001	温度传感器	KY-021	迷你簧片开关
KY-002	震动传感器	KY-022	红外线传感器 / 接收器
KY-003	霍尔磁力传感器	KY-023	双轴 XY 摇杆
KY-004	按键开关	KY-024	线性磁力霍尔传感器
KY-005	红外发射器	KY-025	簧片开关模块
KY-006	无源蜂鸣器	KY-026	火焰传感器
KY-008	激光头传感器	KY-027	魔术光杯模块
KY-009	三色全彩 LED	KY-028	温度传感器
KY-010	光遮断器	KY-031	敲击传感器
KY-011	双色 LED	KY-032	避障传感器
KY-012	有源蜂鸣传感器	KY-033	寻线传感器
KY-013	模拟温度传感器	KY-034	七彩自动闪烁 LED
KY-015	温度湿度传感器	KY-035	霍尔磁性传感器
KY-016	三色 LED	KY-036	金属触摸传感器
KY-017	水银开关	KY-037	高感度话筒
KY-018	LDR 模块	KY-038	话筒传感器
KY-019	5 V 继电器	KY-039	LED 心跳传感器
KY-020	倾斜开关	KY-040	旋转编码器

SainSmart 是另外一家生产传感器模块的厂商。表 9-5 列出了 SainSmart 模块套装中的各种模块。表 9-8 给出了各模块的图片、说明及引脚图，其中列出的套装内容只是一个代表性的例子，具体套装包含的模块可能有所不同。带有 SainSmart 产品编号的模块也可以单独购买，其中一些是独一无二的。

表9-5：SainSmart模块套装

产品编号	名称
N/A	继电器模块
20-011-985	触摸传感器
20-019-100	超声波距离传感器 HC-SR04
20-011-984	火焰传感器
20-011-986	温度与相对湿度传感器
N/A	有源蜂鸣器
20-011-982	震动 / 敲击传感器
N/A	无源蜂鸣器
20-011-987	跟踪传感器

（续）

产品编号	名称
20-011-983	霍尔效应传感器
20-011-981	光敏传感器
N/A	红外接收器
20-011-944	摇杆模块
20-011-946	水浸传感器

最后还有 TinkerKit 模块。表 9-6 列出了常见的 TinkerKit 模块，表 9-9 给出了 Pro 套装中每个模块的细节。TinkerKit 系列模块被设计为与 TinkerKit 接口扩展板（第 8 章已经介绍过）一起使用。尽管目前 TinkerKit 公司处于不稳定状态，但仍然可以从 Mouser（http://www.mouser.com）、Newark/Element14（http://www.newark.com）以及其他渠道买到其产品。相应软件库也可以从 GitHub（https://github.com/TinkerKit）下载使用。在 Mouser（http://bit.ly/mouser-tinkerkit）上，你可以找到一套模块的基本数据手册。

表 9-9 并未提供模块的引脚图，因为 TinkderKit 模块全部使用标准引脚。模块间的唯一区别是，它是离散数字（on/off）模块还是模拟模块。两种类型的模块都使用一样的基本连接器引脚，如图 9-9 所示。

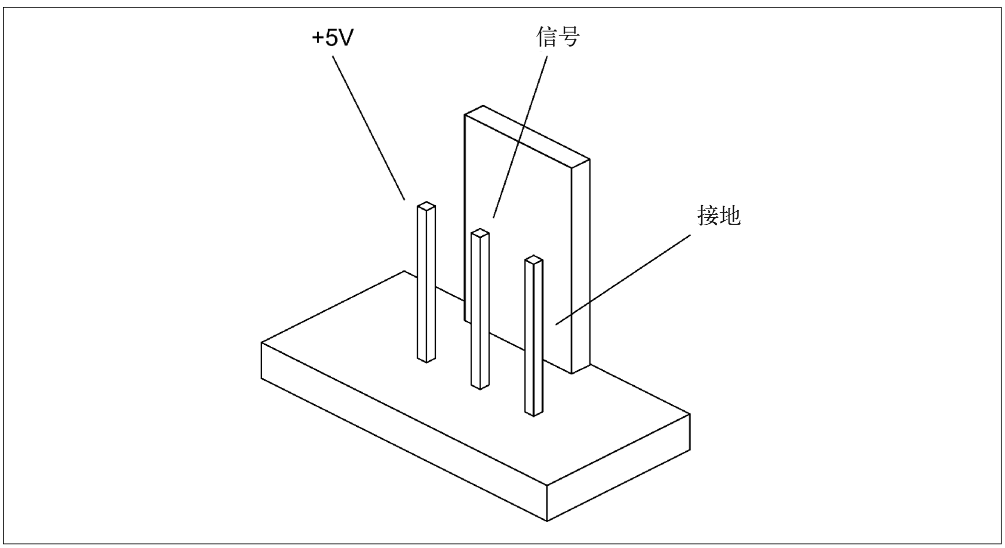


图 9-9：TinkerKit 模块常用连接器

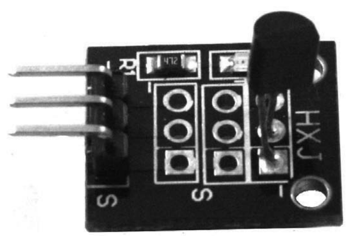
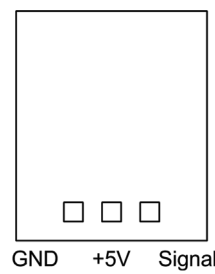
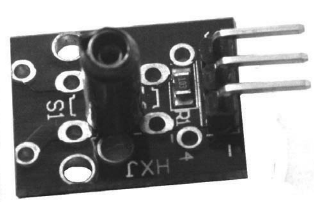
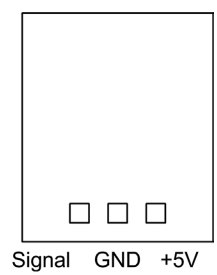
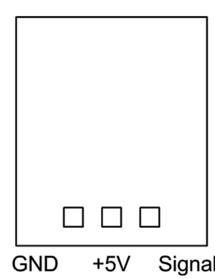
TinkerKit 模块上的电路控制电气接口可能是水平传感（level sensing）、放大（amplification）等，所以请记住，在许多 TinkerKit 模块的协助下，Arduino 并不直接与传感器、LED、输入控制或输出设备进行通信，而是接口电路。请把模块翻过来，检查其背面，看看它安装了什么电路。你会发现一些 LED 模块甚至安装了驱动晶体管。TinkerKit 目标是制造容易连接到 Arduino 的产品，并且这些产品对小差错相对不敏感。为此，他们设计产品时必须放弃一些低级接口的交互能力。

表9-6：TinkerKit模块清单

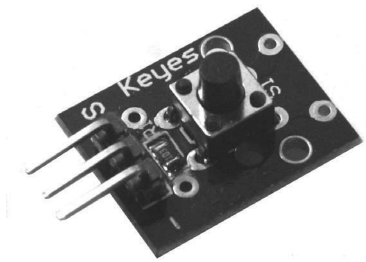
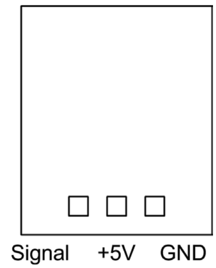
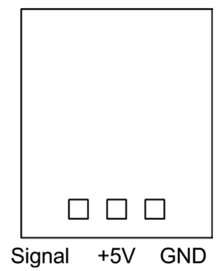
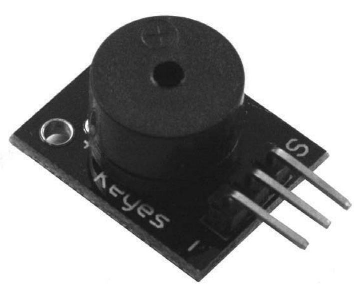
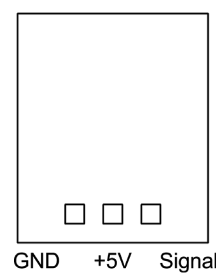
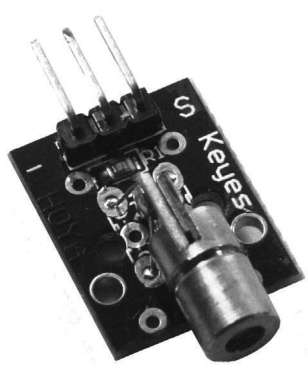
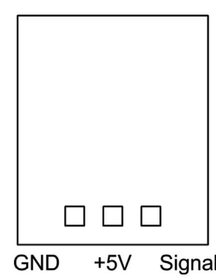
产品编号	名称	产品编号	名称
T000020	加速度感应模块	T010020	场效应管模块
T000030	摇杆模块	T010110	大功率 LED 模块
T000070	霍尔效应传感器	T010111	5 mm 蓝色 LED 模块
T000090	LDR 模块	T010112	5 mm 绿色 LED 模块
T000140	旋转电位器模块	T010113	5 mm 黄色 LED 模块
T000150	线性电位器模块	T010114	5 mm 红色 LED 模块
T000180	按键模块	T010115	10 mm 蓝色 LED 模块
T000190	倾斜模块	T010116	10 mm 绿色 LED 模块
T000200	热敏电阻模块	T010117	10 mm 黄色 LED 模块
T000220	触摸传感器模块	T010118	10 mm 红色 LED 模块

1. KEYES模块

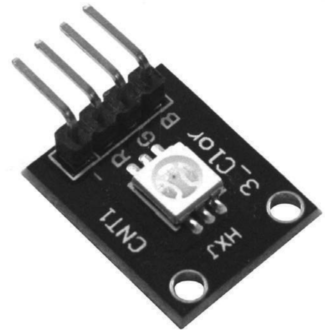
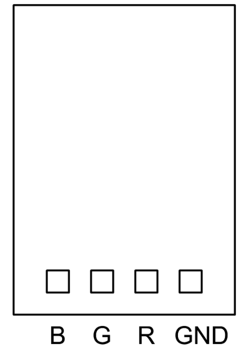
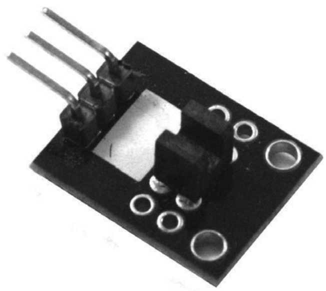
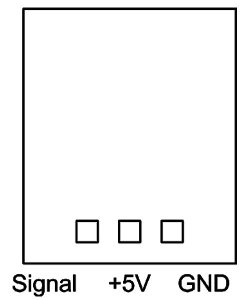
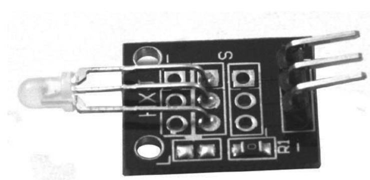
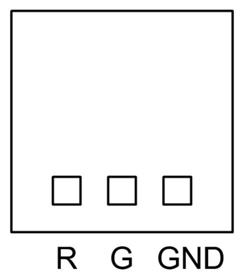
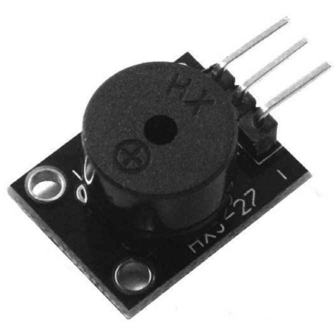
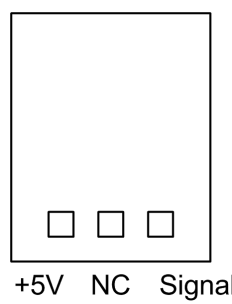
表9-7：KEYES传感器与输出模块

产品编号	名称与描述	图片	引脚
KY-001	温度传感器 采用 DS18B20 单线温度传感器 IC，T0-92 封装		
KY-002	震动传感器 在 GND 与信号引脚之间，一个密封的震动传感器将电路闭合起来。好像未连接 +5 V，但 PCB 底面有一个电阻位。该传感器非常灵敏		
KY-003	霍尔效应磁力传感器 该传感器用于检测有无磁场。输出是 A3144 霍尔效应设备中 NPN 晶体管集电极开路。当传感器激活时，输出被拉至底端。该传感器不是线性的，而 KY-024 是		

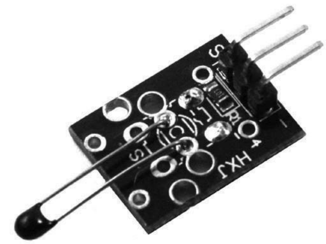
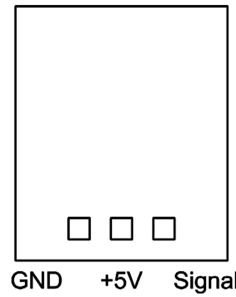
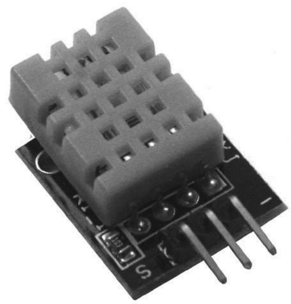
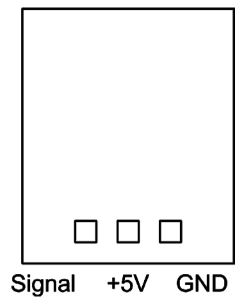
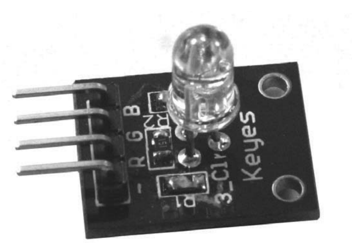
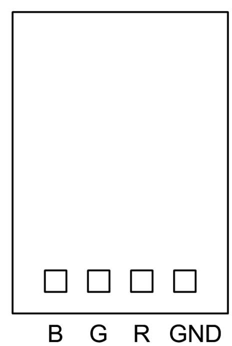
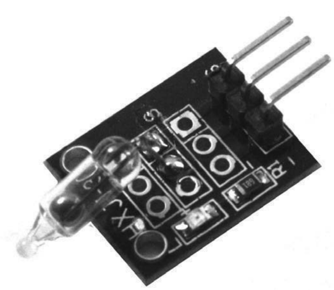
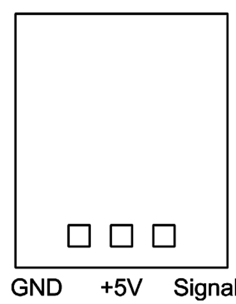
（续）

产品编号	名称与描述	图片	引脚
KY-004	按键开关 一个简单的按键开关。当开关按下时，输出被下拉（Pull Low）		
KY-005	红外线发射器 红外线 LED，与 KY-022 配合使用。请注意，使用时必须使用外部限流电阻，且不要将 GND 引脚连接到测试模块		
KY-006	无源蜂鸣器 这是一个带有金属膜片的小喇叭。未使用 +5 V 引脚		
KY-008	激光 LED 低功耗 650 nm（红色）LED 激光（见图 9-51）。GND 连接到阳极，信号引脚连接到阴极。未使用 +5 V（但通过 PCB 上的一个 10K 电阻连接到信号端）。该模块需要一个外部限流电阻		

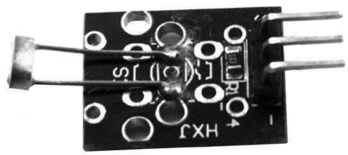
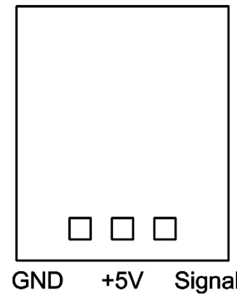
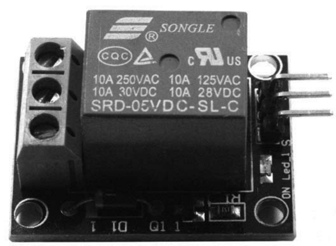
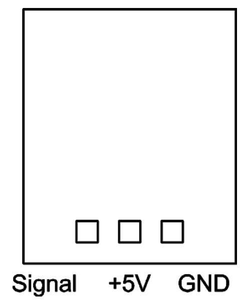
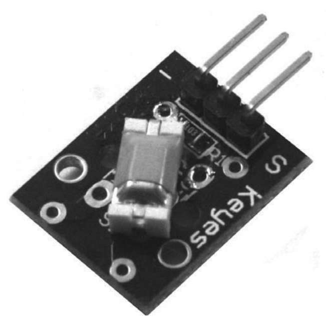
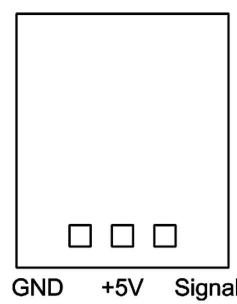
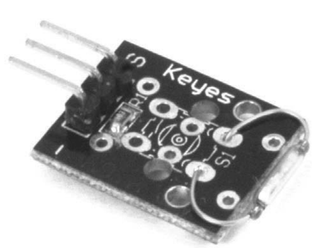
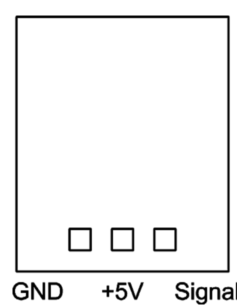
（续）

产品编号	名称与描述	图片	引脚
KY-009	三色全彩 LED 采用一个同时输出红、绿、蓝的 LED。测试过几个模块发现有引脚不一致的情况，使用前请先检查每种颜色是如何连接的		
KY-010	光遮断器 包含一个光遮断器（一个 LED 与光敏晶体管），用于检测缝隙之间是否有物体		
KY-011	双色 LED 采用能够输出绿色或红色，或同时输出两者的 LED。它是一个三导线（three-lead）器件，这意味着内部 LED 公共一个连接，每个都能独立运行		
KY-012	有源蜂鸣器 通电时产生固定音高		

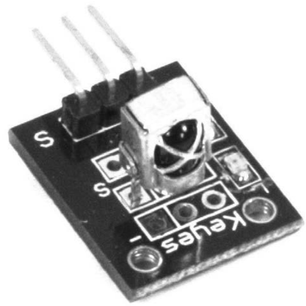
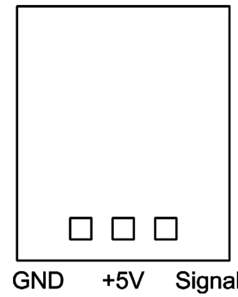
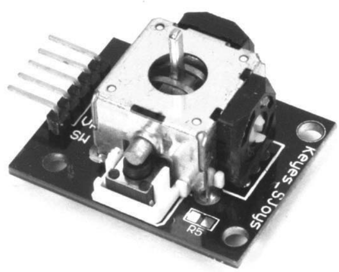
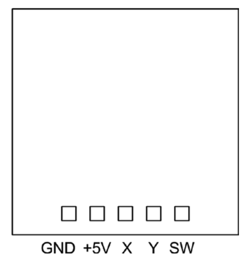
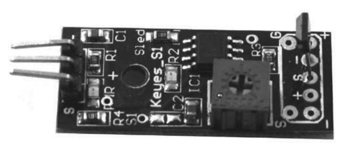
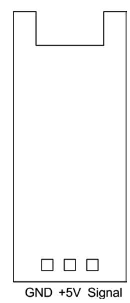
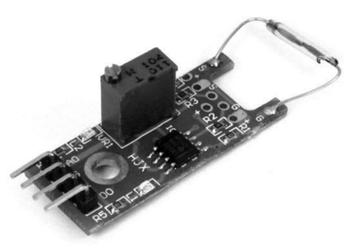
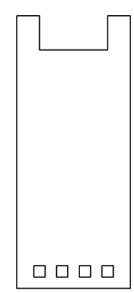
（续）

产品编号	名称与描述	图片	引脚
KY-013	模拟温度传感器 采用热敏电阻（见 9.4.1 节中的“热敏电阻”部分）作为温度感应元件。输出电压是温度的函数		
KY-015	温度与湿度传感器 采用 DHT11 温度与湿度传感器。更多信息请参考 9.4.1 节中的“DHT11 与 DHT22 传感器”部分		
KY-016	三色 LED 基本与 KY-009 相同，但采用过孔部件（throug-hole）代替表面贴装（surface-mount）设备		
KY-017	水银倾斜开关 借助玻璃管中的一小滴水银，检测在一个轴上是否发生倾斜（请参考 9.4.2 节中的“单轴倾斜传感器”部分）。要感知两个轴上的倾斜，则要使用两个模块。+5 V 引脚未连接到传感器，但经由 LED 与 680 欧姆电阻连接到信号引脚		

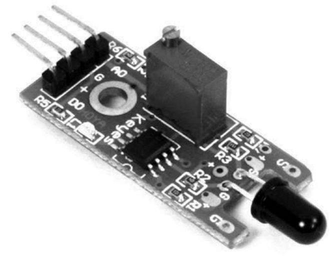
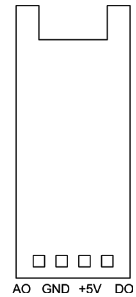
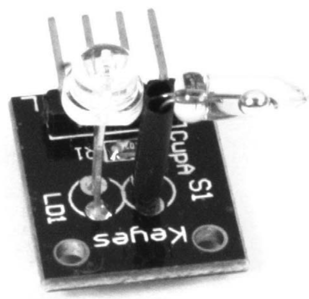
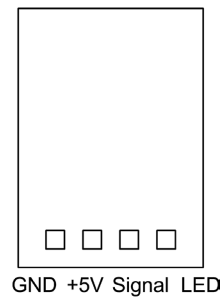
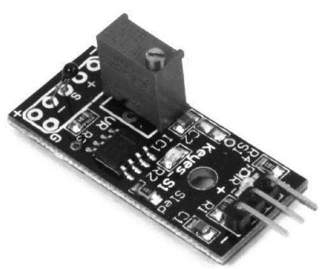
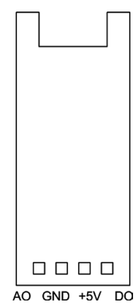
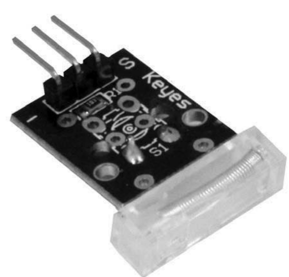
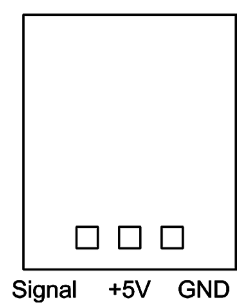
（续）

产品编号	名称与描述	图片	引脚
KY-018	LDR 模块 光敏电阻（light-dependent resistor，简称 LDR）模块。输出电压随着照射到光敏电阻上光线的数量而变化（请参考 9.4.4 节）		
KY-019	5 V 继电器 一个小的安装在 PCB 上的继电器（请参考 9.6.2 节中的“继电器”部分），带有内置驱动器，连接至逻辑电平控制信号		
KY-020	倾斜开关 类似于 KY-017，但采用的是包含于小封闭空间的金属球		
KY-021	迷你磁簧开关 一个封装于小玻璃管的磁簧开关，遇到磁场时闭合		

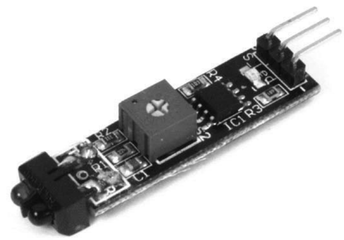
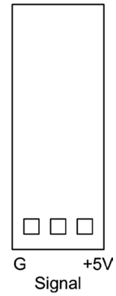
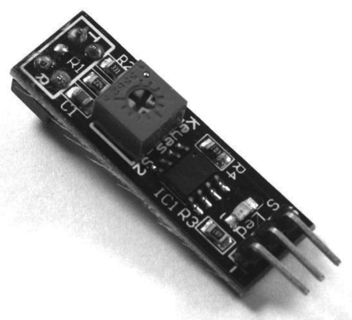
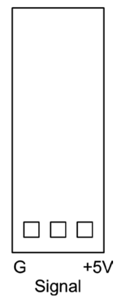
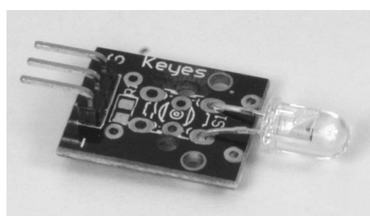
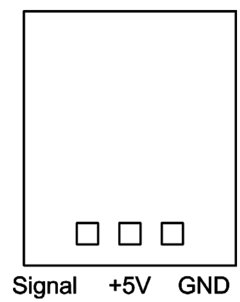
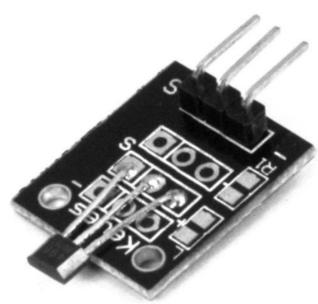
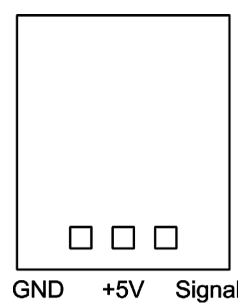
（续）

产品编号	名称与描述	图片	引脚
KY-022	红外线传感器 / 接收器 这类 1838IR 传感器用于远程控制电视以及其他电器。运行频率为 37.9 kHz，出现 IR 载波会引起高电平输出		
KY-023	双轴 XY 摇杆 包含两个电位器，它们成直角安装，以便检测轴中心的 x、y 移动		
KY-024	线性霍尔效应传感器 采用 SS49E 线性霍尔效应传感器。一个 LM393 电压比较器与电位器一起使用，以便调整电路的灵敏度		
KY-025	磁簧开关模块 采用一个磁簧开关与比较器电路（见图 9-8）		

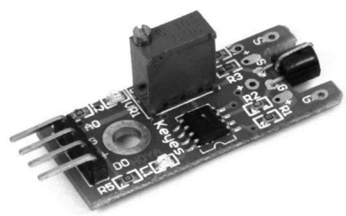
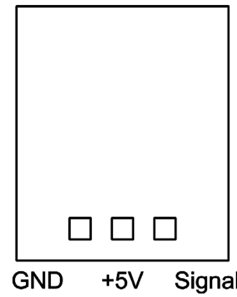
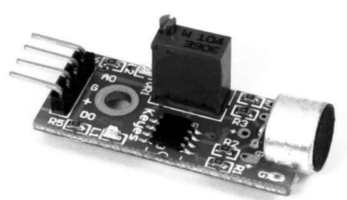
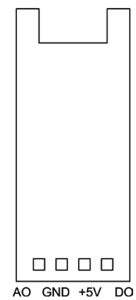
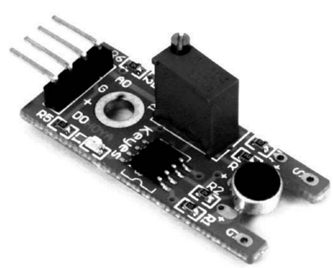
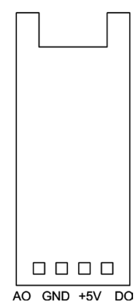
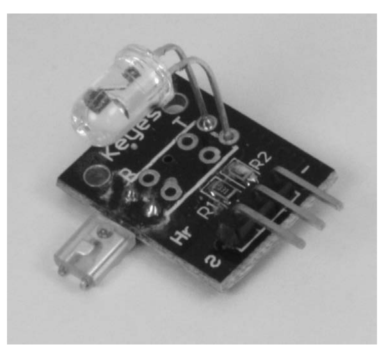
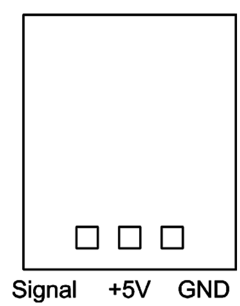
（续）

产品编号	名称与描述	图片	引脚
KY-026	火焰传感器 采用一个优化的 IR 传感器，波长 760 nm~1100 nm。电位器用于设置灵敏度		
KY-027	魔术光杯模块 坦率地说，我不知道这个模块用来干什么。 KY-027 本质上是在独立电路上带有一个 LED 的 KY-017		
KY-028	温度传感器 使用一个热敏电阻与一个比较器 IC 检测阈值（请参考 9.4.1 节中的“热敏电阻”部分）。电位器用于设置阈值		
KY-031	敲击（碰撞）传感器 用于检测剧烈碰撞，并产生输出。它并不像 KY-002、KY-017、KY020 一样检测倾斜		

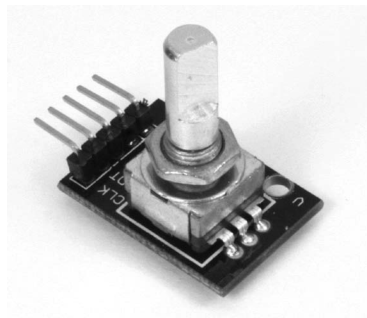
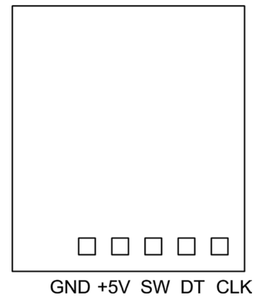
（续）

产品编号	名称与描述	图片	引脚
KY-032	IR 接近传感器 采用 IR 光源反射检测附近表面或障碍物。电位器用于设置灵敏度。请注意，这个模块至少有两个变种。通电前请先检查引脚		
KY-033	IR 寻线传感器 使用来自一个表面的 IR 光线的反射，检测明暗差异。通常用于制作寻线机器人，在白色地面上跟踪黑线。电位器用于设置灵敏度。本质上，该模块与 KY-032 一样，但它带有 LED 与 IR 传感器，它们安装在 PCB 底面		
KY-034	七彩自动闪烁 LED 通电时，LED 将自动闪烁		
KY-035	霍尔效应磁力传感器 类似于 KY-003，但带有 SS49E 线性霍尔效应传感器		

（续）

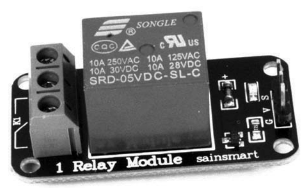
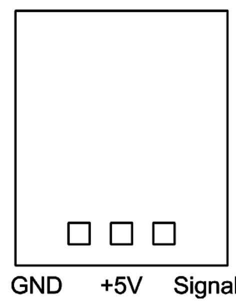
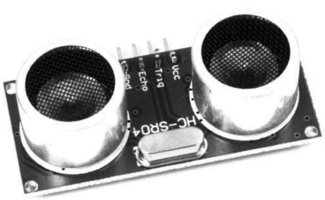
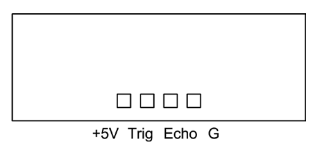
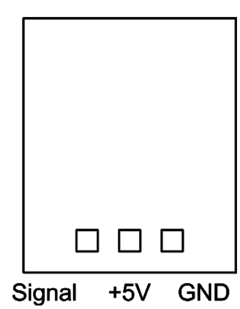
产品编号	名称与描述	图片	引脚
KY-036	导电接触传感器 当导电物质（比如手指）接触到传感器的裸露导线时，输出将发生变化		
KY-037	高感度话筒传感器 话筒带有可变阈值。话筒越大（相比于 KY-038），该模块越敏感		
KY-038	话筒传感器 话筒带有可变阈值。由于该模块话筒较小（相比于 KY-037 模块），所以不太灵敏		
KY-039	LED 心跳传感器 类似于医院或诊所的医生用手指为病人测脉搏（心跳）。使用光敏晶体管检测来自 LED 光线的轻微变化，就像血液流经手指时感受到得那样		

（续）

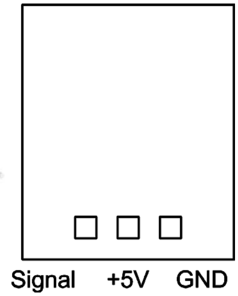
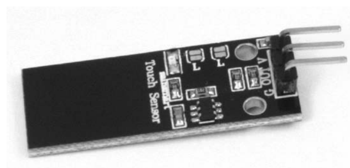
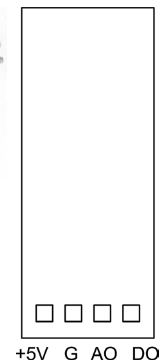
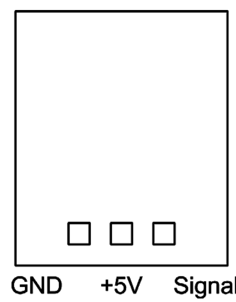
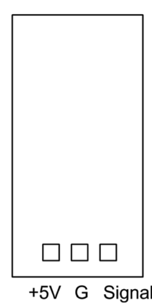
产品编号	名称与描述	图片	引脚
KY-040	旋转编码器 数字连续式旋转编码器。更多相关内容请参考 9.4.8 节中的“数字旋转编码器”部分		

2. SainSmart模块

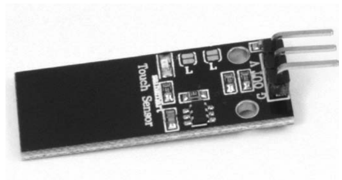
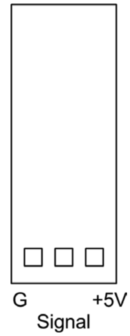
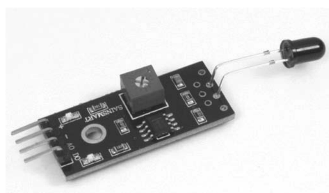
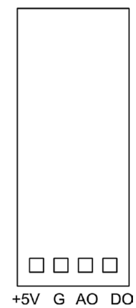
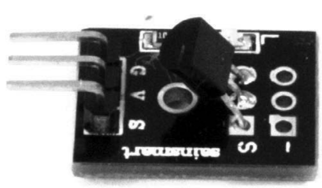
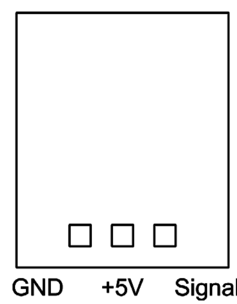
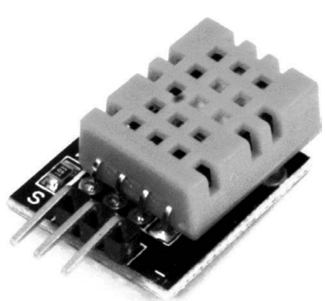
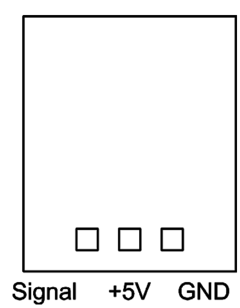
表9-8：SainSmart传感器与输出模块

产品编号	名称与描述	图片	引脚
	继电器模块 带有 10 A 接触电流的 5 V 继电器		
20-019-100	超声波距离传感器 HC-SR04 使用一对超声波换能器发射信号，并接收回声。超声波从发射到接收所需的时间和传感器与反射面之间的距离成正比		
	有源蜂鸣器 通电时发声		

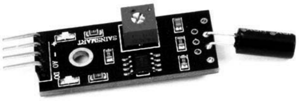
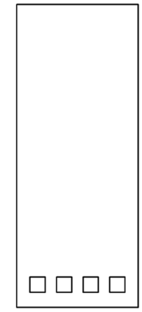
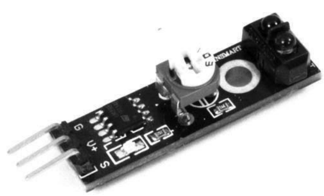
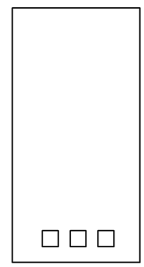
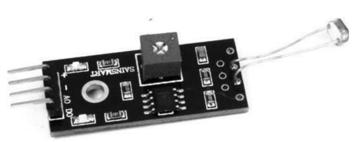
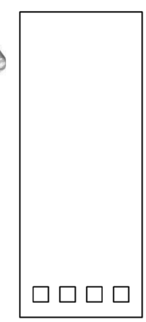
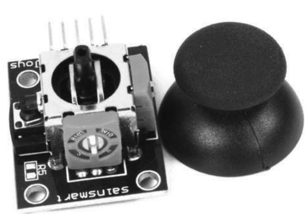
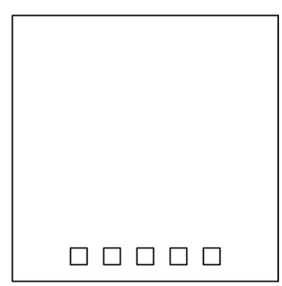
（续）

产品编号	名称与描述	图片	引脚
	无源蜂鸣器 响应方波输入，它允许该模块发出一个可编程音调		
20-011-983	霍尔效应传感器 采用一个线性霍尔效应传感器。小电位器用于设置灵敏度阈值		
	红外线接收器 响应来自远程控制设备的脉冲，并产生数字信号		
20-011-946	水浸传感器 通过响应 PCB 上“金属手指”（metallic finger）间导电率的变化检测有无水浸。也可以用作一个雨水探测器或泼溅探测器		

（续）

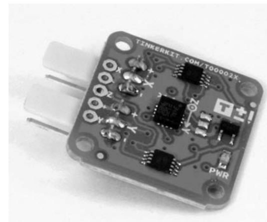
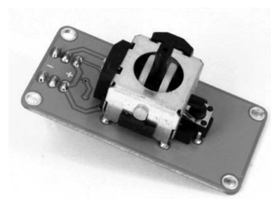
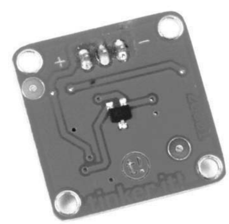
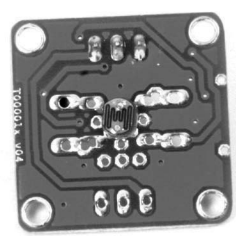
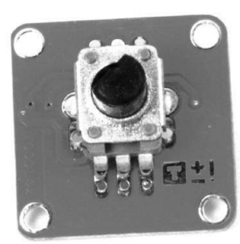
产品编号	名称与描述	图片	引脚
20-011-985	触摸传感器 检测是否有手指触摸		
20-011-984	火焰传感器 响应波长 760 nm~1100 nm 的 IR。电位器用于设置灵敏度		
20-011-988	温度传感器 使用 DS18B20 单线温度传感器 IC（请参考 9.4.1 节 DS18B20 部分），采用 T0-92 封装方式		
20-011-986	温度和相对湿度传感器 采用 DHT11 温度与湿度传感器。有关该元件的更多内容，请参考图 9-14		

（续）

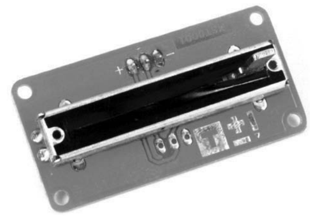
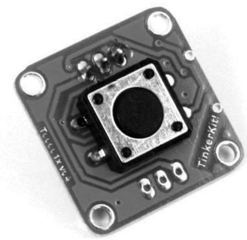
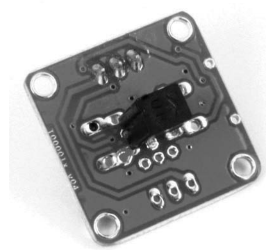
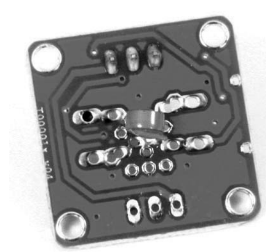
产品编号	名称与描述	图片	引脚
20-011-982	震动 / 敲击传感器 感知突然运动（敲击或震动），带有密封传感器，类似于 KEYES KY-002		
20-011-987	跟踪传感器 采用一个 IR LED 与光敏传感器，检测反射率		
20-011-981	光敏传感器 一个基本的 LDR 电路（见 9.4.4 节），且灵敏度可调		
20-011-944	摇杆模块 两轴线性电位器摇杆，包括一个大旋钮		

3. TinkerKit模块

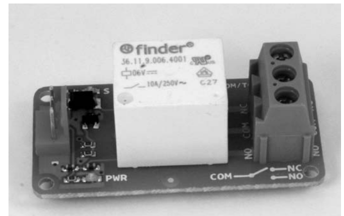
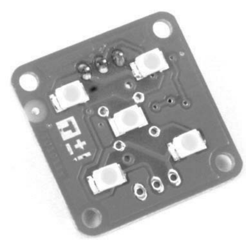
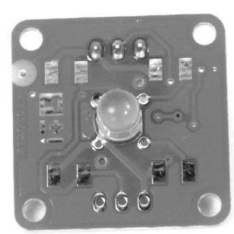
表9-9：TinkerKit I/O模块

产品编号	名称与描述	图片
T000020	加速度传感器模块 基于三轴 LIS344AL IC，包括两个信号放大器	
T000030	摇杆模块 两个电位器，安装于双轴万向节上	
T000070	霍尔效应传感器 根据局部磁场强度输出电压	
T000090	LDR 模块 带有放大器的光敏电阻，输出电压与光水平成正比	
T000140	旋转电位器模块 4.7 Ω 旋转模拟电位器	

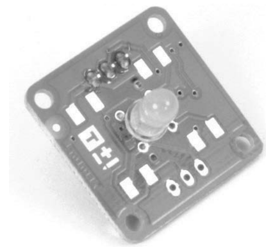
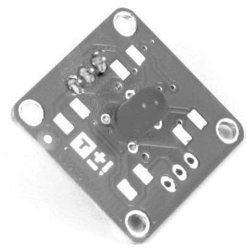
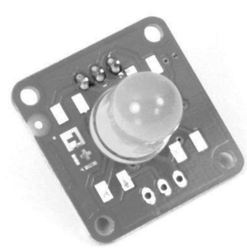
（续）

产品编号	名称与描述	图片
T000150	线性电位器模块 4.7 Ω 线性（滑块）模拟电位器	
T000180	按键模块 简单的按键模块，常开，按下时输出 +5 V	
T000190	倾斜模块 采用简单的带有内部金属球的倾斜传感器	
T000200	热敏电阻模块 采用热敏电阻与放大器，输出电压与温度成正比	
T000220	触摸传感器模块 采用 QT100A 单触摸控制器，触摸传感器时产生 5 V 电压。触摸板是 PCB 前面的一部分，在图像中表现为一个平面	

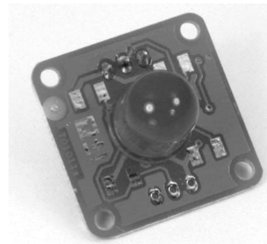
（续）

产品编号	名称与描述	图片
T010010	继电器模块 集成一个 5 V 继电器（触点耐压为 250 V 10 A）、激励晶体管、螺旋式端子台	
T010020	场效应管模块 采用 IRF520 MOSFET 切换到 24 V DC。速度快，可以与 PWM 一起使用控制 DC 电动机	
T010110	大功率 LED 模块 提供 5 个超亮 LED。需要大电流，所以可能需要使用 T010020 与外部电源供给	
T010111	5 mm 蓝色 LED 模块 一个简单的模块，带有一个 5 mm 蓝色 LED	
T010112	5 mm 绿色 LED 模块 一个简单的模块，带有一个 5 mm 绿色 LED	

（续）

产品编号	名称与描述	图片
T010113	5 mm 黄色 LED 模块 一个简单的模块，带有一个 5 mm 黄色 LED	
T010114	5 mm 红色 LED 模块 一个简单的模块，带有一个 5 mm 红色 LED	
T010115	10 mm 蓝色 LED 模块 一个简单的模块，带有一个 10 mm 蓝色 LED	
T010116	10 mm 绿色 LED 模块 一个简单的模块，带有一个 10 mm 绿色 LED	
T010117	10 mm 黄色 LED 模块 一个简单的模块，带有一个 10 mm 黄色 LED	

（续）

产品编号	名称与描述	图片
T010118	10 mm 红色 LED 模块 一个简单的模块，带有一个 10 mm 红色 LED	

9.2 Grove模块

SeeedStudio 已经积累了大量 Grove 互联系统模块，其中许多模块的功能与 KEYES、SainSmart、TinkerKit 模块类似，而其他一些模块则是 Grove 产品线所特有的。

按照不同类别，可以将 Grove 模块划分为 6 组，分别为环境监测、运动感知、用户接口、物理监测、逻辑门功能、电源控制。所有模块使用标准接线方案，包含一个四引脚的连接器模块，带有电源线、地线、信号线。

与 TinkerKit 模块一样，Grove 模块的优点是，你不必尝试核实（或弄清）模块的引脚实际做什么。通过一条预制电缆即可轻松地将 Grove 传感器或驱动器模块连接到遵循 Grove 约定的扩展板（比如第 8 章中介绍的 Grove 扩展板）上。Seeed Studio 也销售预制电缆，借助这些电缆，可以把各种模块连接到一个接口或控制板上。

Grove 模块一个很吸引人的特征是，大部分模块的 PCB 都包含安装支架。这使得安装硬件不需要使用复杂电路（与 PCB 中间的安装孔相反），连接器模块也很容易使用。一个 Grove 模块的例子如图 9-10 所示。

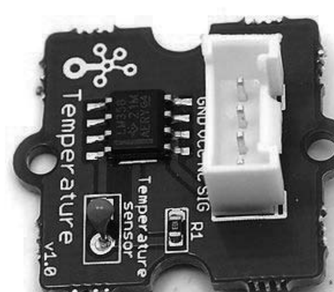


图 9-10 ：一个 Grove 模块的示例

写作本书之时，TinkerKit 模块看起来正在消失，而 Grove 模块似乎十分活跃、生机勃勃。然而，这并不意味着 TinkerKit 不会卷土重来，也不代表 Grove 会长盛不衰。在 Arduino 世界，短时间内事情就可能发生翻天覆地的变化。正如我之前指出的，本章与第 8 章只展示

一些可用的模块、元件，供你参考，但不具权威性。市场变幻莫测，未来的事谁也说不好。

如果你正在寻找遵循一种或另一种约定的模块，那么我建议研究 Grove 的产品。如果想使用一种不同的扩展板或接口板，你可以购买壳体、引脚、插口制作自己的接口电缆，具体内容请参考 9.10.3 节。在 Seeed Studio wiki（http://bit.ly/seeed-grove）上，你可以看到可用模块的完整列表。

9.3 传感器与模块介绍

接下来将介绍一些不同类型且与 Arduino 兼容的元件、模块与传感器。我之所以说“一些”是因为，它们与扩展板的情况一样，市场上会不断推出新的传感器与模块，同时一些旧的会消失，因为它们没有占有足够的市场份额以便使自己继续存活下去。一些模块专用性很强，仅有一种类型，比如煤气传感器、电流监测器。一些传感器可能会因新型传感器将其取代而消失。使用谷歌搜索引擎浏览 Amazon.com 商品清单，或查看 9.11 节列出的资源网站，几乎不费什么力气就能找到此处介绍的所有设备，甚至更多。

表 9-10 列出了本部分将要介绍的控制器、传感器、驱动器、显示器、模块。它们按照功能、类别（输入或输出）、类型（设备、元件、模块）组织起来。请记住，模块上的各种元件也可以单独购买，所以你可以自己制作模块，或者把它们集成到其他地方。

表9-10：传感器与模块列表

功能	类别	类型	说明
声音	输出	设备	蜂鸣器（报警器）元件与模块
声音	传感器	话筒	拾音器模块（话筒）
通信	I/O	模块	315/433 MHz RF 模块
通信	I/O	模块	APC220 无线模块
通信	I/O	模块	ESP8266 Wi-Fi 收发器
通信	I/O	模块	NRF24L01 模块
接触	传感器	开关	接触式开关模块
接触	输出	开关	继电器和继电器模块
控制	输入	设备	键盘
控制	输入	模块	摇杆模块
控制	输入	设备	电位器
显示	输出	模块	七段模块
显示	输出	模块	ERM1601SBS-2 16 × 1 LCD 显示器
显示	输出	模块	ST7066 (HD44780) 16 × 2 LCD 显示器
显示	输出	模块	ERC240128SBS-1 240 × 128 LCD 显示器
显示	输出	模块	ST7735R 128 × 160 TFT 显示器
发光	输出	显示器	七段 LED 显示器
发光	输出	显示器	LED 矩阵模块
发光	输出	激光	激光 LED
发光	输出	LED	单色 LED

（续）

功能	类别	类型	说明
发光	输出	LED	双色 LED
发光	输出	LED	三色（RGB）LED
光线感知	传感器	光敏电阻	LDR 模块
光线感知	传感器	二极管	光电二极管模块
光线感知	传感器	晶体管	光电晶体管模块
IR 感知	传感器	IR	PIR 传感器模块
磁力	传感器	固态器件	霍尔效应传感器模块
磁力	传感器	固态器件	磁强计模块
湿度	传感器	PCB	土壤湿度传感器模块
运动	输出	驱动器	DC 电机控制
运动	输出	驱动器	舵机控制
运动	输出	驱动器	步进电机控制
运动	传感器	固态器件	陀螺仪模块
运动	传感器	固态器件	加速度传感器模块
压力	传感器	固态器件	大气压力传感器模块
距离	传感器	模块	激光发射与接收模块
距离	传感器	模块	LED 目标传感器模块
距离	传感器	模块	超声波测距仪模块
旋转	传感器	控制	数字旋转编码器模块
信号	输出	模块	波形产生器模块
温度	传感器	固态器件	DS18B20 温度传感器模块
温度	传感器	模块	DHT11/DHT22 温度与湿度传感器模块
温度	传感器	模块	热敏电阻温度传感器模块
倾斜	传感器	开关	单轴倾斜传感器模块
倾斜	传感器	开关	双轴倾斜传感器模块
时间	协助	模块	DS1302 RTC 模块
时间	协助	模块	DS1307 RTC 模块
时间	协助	模块	DS3231 RTC 模块
时间	协助	模块	PCF8563 RTC 模块
电压	输出	模块	DAC 模块
水分	传感器	PCB	水电导率传感器

9.4　传感器

尽管我们可以直接将各种传感器连接至 Arduino，但上面列出的各种模块使用起来显然会更容易。然而，如果你想为某个特定应用制作硬件，那么使用模块可能不是一个好的选择。这种情况下你将只想使用传感器组件，并且希望把它们准确放置到需要的位置。

传感器是一个恒输入设备，它从物理环境获取数据，并将其转换为微控制器可以处理的形式。顾名思义，传感器是指可以感知“事物”的设备，感知的对象可能是温度、湿度、磁场、可见光、热量、红外线、声音或物理运动。

9.4.1　温度、湿度、压力传感器

市面上有许多环境传感器（environmental sensors）可供选择，它们都能与 Arduino 一起使用。从简单基于连续性的水浸传感器到灵敏的温湿度传感器，总会有一些传感器可以用于测量环境的冷热与干湿程度。

1. DS18B20

DS18B20 是所谓的单总线（one-wire）温度传感器，它返回二进制数据流，其中包含着传感器感知到的当前温度。图 9-11 展示了一个常用的 DS18B20 模块，它由一个 DS18B20、一些无源元件以及一个 LED 组成。

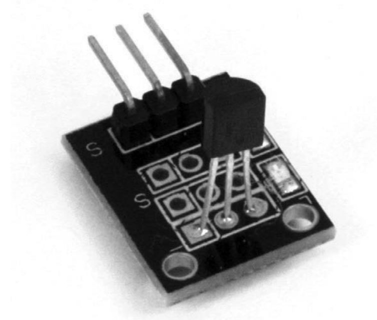


图 9-11：常见的 DS18B20 模块

图 9-12 展示了 DS18B20 模块安装于一个原型开发板上的情形，该原型开发板同时安装有一个继电器与一个电位器。顺便说一下，这是一个简单的数字恒温器，将其拼凑起来用于取代小型便携式电暖器中的双金属片式机电恒温器。旧式的机械式恒温器总是不能很好地运行，但这个简单的数字替代品却能出色地完成任务。并且它所连接的 Arduino 也有能力记录温度数据，以及执行其他功能。接入物联网的一个便宜的电暖器？当然，为什么不？

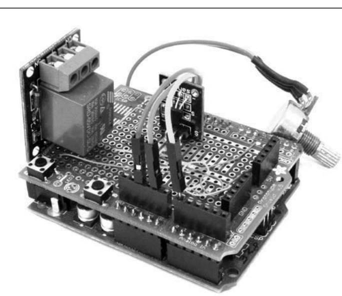


图 9-12：DS18B20 模块应用示例

2. DHT11与DHT22传感器

DHT11 与 DHT22 温度与湿度传感器采用三端塑料封装方式，如图 9-13 所示。你也会看到它们安装在小型 PCB 上，如图 9-14 所示。

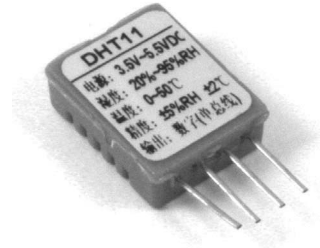


图 9-13：DHT11 封装

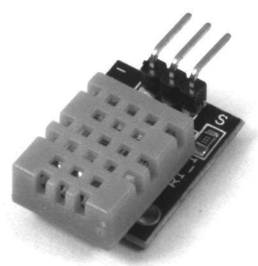


图 9-14：DHT11/DHT22 模块

从分辨率与串行数据速率看，DHT11 与 DHT22 传感器不同。DHT11 是一个基本设备，其温度感知精度为 +/−2℃，湿度分辨率为 +/−5% 相对湿度，工作范围为 20%~90% 相对湿度、0℃ ~50℃。DHT22 温度分辨率为 +/−0.2℃，湿度感知精度为 +/−1% RH，拥有更快的串行数据定时速率（timing rate）。与 DHT11 相比，DHT22 感知范围更广：−40℃~80℃、0%~100% RH。DHT22 的引脚与 DHT11 完全相同。

DHT11 与 DHT22 都采用非标准的单总线串行通信协议，使用信号响应方案（signal-response approach）。微控制器先暂时拉低单信号线，然后允许它拉高（借助上拉电阻，可以是外部添加的，也可以是 PCB 模块上的）。DHT11/ DHT22 使用 40 位的串行数据做响应，由 5 个 8 位数据值组成，其中包含一个 8 位的校验和。

3. 热敏电阻

热敏电阻是温度控制电阻器，有正负温度系数可用。随着温度升高，负温度系数（NTC）设备的电阻会变小，而正温度系数（PTC）设备则表现出相反的行为。NTC 热敏电阻是应用在传感器中最常见的类型，PTC 类型常常被用作浪涌电流限制器（current inrush limiters）。

图 9-15 显示如何将热敏电阻连接到 Arduino，但请注意，热敏电阻的响应曲线并不是线性的。一些电路使用恒流源代替固定电阻。

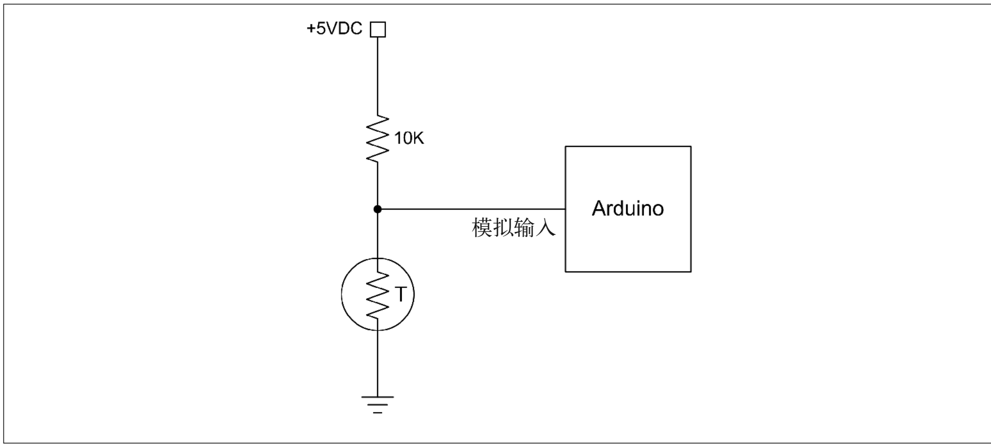


图 9-15：连接到 Arduino 的简单热敏电阻

尽管可以直接将热敏电阻连接到 Arduino，但更简单的方法是使用模块。该模块包含必要的无源元件，便于为热敏电阻创建分压器，比如表 9-7 中的 KY-028 模块。也可以使用图 9-26 显示的运算放大器电路（op amp circuit），以提高热敏电阻的灵敏度。请注意，由于大部分用于测量温度的热敏电阻是 NTC 类型的，所以 Arduino 模拟输入上的电压会随着温度的升高而降低。

4. 水浸传感器

水浸传感器十分有用，可以应用于许多场合，从地下室的洪水传感器到自动气象站的雨水探测器。表 9-8 展示了一种来自 SainSmart（20-011-946）的水浸传感器。这款传感器集成了一个 NPN 晶体管，当 PCB 上的薄薄的迹线被水滴连通时，或者湿物引起晶体管导通时，NPN 晶体管就会拉低输出。

图 9-16 显示了水浸传感器的电路图。如你所见，水浸传感器并不复杂，几乎可以与任何导线或探头一起使用。将这个电路连接到一对钢丝，然后把钢丝安装在地下室地板上，使其向上露出 1/4 in（大约 1 cm）左右。当洪水开始淹没地下室时，它将触发楼上的报警器。

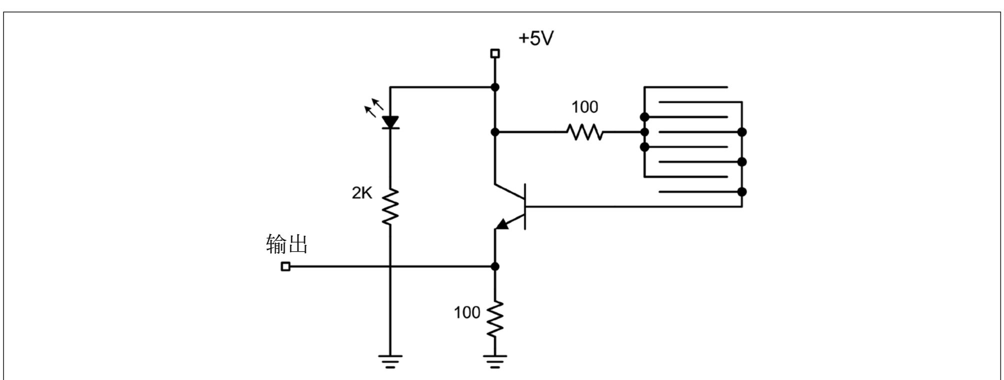


图 9-16：水浸传感器电路图

5. 土壤湿度传感器

简单来说，带有两只插脚的土壤湿度传感器其实只是个电导探头。探头被配置为充当简单分压器中的一个组件，其上的电压是探头之间土壤电导率的函数。可以从供应商（比如 SainSmart）那里购买一个套件，它由一个水分探头、一个小接口模块以及电缆组成。图 9-17 显示了这样一种套件，它由 3 部分组成。

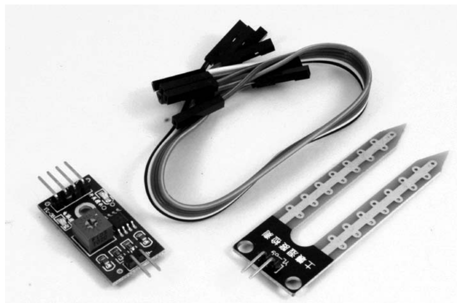


图 9-17：带有接口模块的土壤水分探头套件

针对于探头，几乎使用任何导电材料都能获得同样的效果。在那些存在腐蚀的场合下，为了能够长期使用，选用不锈钢或碳棒等材料可能是更好的选择。某种类型的放大器或缓冲器是必需的，使用它们可以获取一致的读数，而无需将大量电流导向探头（这可能产生一些有趣的副作用，也可能腐蚀探头的电极）。

Grove 101020008 是土壤湿度传感器的另一个变种，它来自 Seeed Studio，如图 9-18 所示。这款传感器包括一个板载 NPN 晶体管，帮助将电压降到 Arduino AVR ADC 能够使用的水平。探头插脚上的铜层有一层薄薄的镀金，用于抵抗腐蚀。

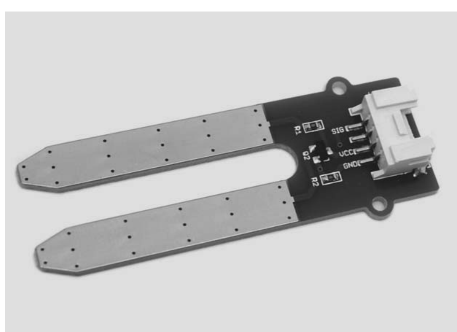


图 9-18：独立的土壤水分探头

图 9-19 显示的是独立的水分探头的电路图。它与水浸传感器中使用的电路在本质上是一样的，如图 9-16 所示。请注意，设计采用了一个四端连接器，取代了图 9-17 中探头所用的引脚连接。很显然，这会更方便，但请注意连接器中各个端子是如何进行连接的。

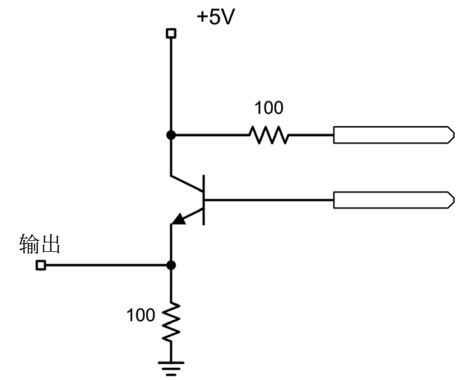


图 9-19：独立土壤水分探头电路图

6. 大气压力传感器

借助于大气压力传感器（如图 9-20）以及前面介绍的 DHT11 或 DHT22 模块（请参考之前的“DHT11 与 DHT22 传感器”部分），使用 Arduino 可以搭建一个小型气象监测器。此处所说的“大气压力传感器”基于 MPL115A2 传感器，并带有一个 I2C 接口。



图 9-20：大气压力传感器模块

这个模块精度并不高，不能用作高度表，但适合获取与记录气象数据。而其他一些基于 MPL3115A2 或 BMP085 传感器的模块精度很高，适合用作高度计。

9.4.2　倾斜传感器

倾斜传感器通常只是一个小的密封管，其中含有一系列内部触点与一个小金属球或汞珠。如果设备发生倾斜，偏离平衡位置（垂直于当地引力方向），金属球或汞珠就会移动，从而使得触点连通，导致电路闭合，其工作原理类似于合上电闸。

请记住，倾斜传感器并不是一个线性传感器（proportional sensor），它要么是倾斜的，要么不是；其电路要么是开放的，要么是闭合的，此外不存在第三种状态。

1. 单轴倾斜传感器

图 9-21 显示了一个带有汞珠的倾斜传感器模块，它就是前面介绍的 KY-017 模块。该模块

只能感知一个方向上的倾斜。感知倾斜时，你需要使用两个或更多个传感器，每个都沿着各个轴端进行排列。

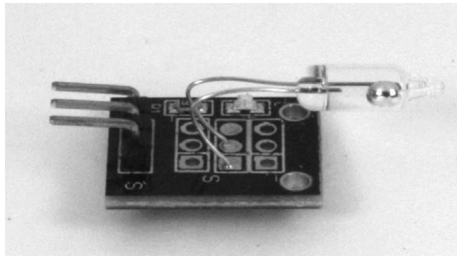


图 9-21：单轴倾斜传感器

2. 双轴倾斜传感器

有效使用倾斜传感器的关键是，确定特定轴上的中间位置（neutral position）。一旦确定，传感器就能沿着某个特定的水平轴感知倾斜。图 9-22 显示了单个基座上的两个倾斜传感器，它们分别可以在 x 轴与 y 轴上感知倾斜。请注意，这种安排将只能在一个方向上感知倾斜（或上或下，具体取决于如何它们是如何安装的）。

如果想在两个坐标轴上感知倾斜（上或下），那么需要好好安排 4 个倾斜传感器，使其互相垂直。这样即可同时在 +/−x 与 +/−y 方向上检测倾斜。与固体压电振动陀螺仪（solid-state gyroscope，比如 9.4.7 节中的“陀螺仪”部分介绍的那些）不同，这种电路不要求有起始参考（starting reference），只要存在引力，它就能工作。缺点是没有中间状态，倾斜传感器要么打开，要么关闭。

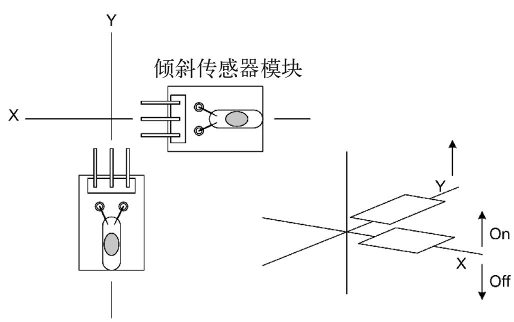


图 9-22：双轴倾斜传感器

9.4.3 声音传感器

话筒有许多有趣的用途。一种是用作安全系统的一部分，用于检测高噪声，比如打碎玻璃的声音，有人踢门的声音，或者枪声。如果话筒灵敏度足够高，甚至可以用于检测脚步声。

接触式话筒可以从处于运行状态的内燃机甚至电动机收集诊断数据。通过这种方式也可以检测噪声水平，以及部件是否发生松动。与光学传感器组合使用时，全向话筒可以用于制作闪电距离探测器（从观察者到闪电的每 1.6 km 距离，闪光发生后大约 4 s 雷声才到达）。

此外，还要有一些小模块可用，它们带有麦克风与 IC，如图 9-23 所示。模块上带有一个小型电位器，用于设置电路的跳变阈值（trip threshold）。该模块使用的电路与图 9-8 所示的电路类似。

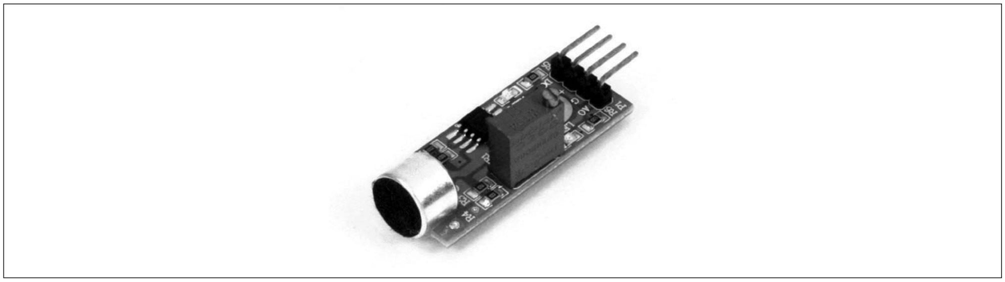


图 9-23：拾音模块

也可以直接将话筒连接到 Arduino 的其中一个模拟输入上，但这样就无法控制话筒的灵敏程度了。一个简单运算放大器电路（如图 9-26 所示）可以提高话筒的灵敏度。

9.4.4 光线传感器

其实，用于探测光线的传感器类型多种多样。有些能够探测热量，比如红外线（IR）传感器；有些能对火焰发出的 IR 做出响应；有些则能对可见光做出响应。一些光线传感器采用了特殊的电阻元件，当有一定数量的光线照射时，电阻自身的阻值就会发生变化。其他一些则采用半导体，用于进一步提高灵敏度以及实现快速响应。

1. 光敏电阻

顾名思义，光敏电阻（light-dependent resistor，LDR）是一种特殊的电阻器，其阻值的变化与照射到其上的光线数量呈现一定的函数关系。大部分光敏电阻的外形如图 9-24 所示。虽然这些设备不如光电二极管与光电晶体管快，但它们的速度足以在一个调幅光束内载送声音。光敏电阻非常有用，环境光探测器、简单低速脉冲编码光数据链路、机器人的信标传感器（协助寻找充电站）以及太阳能电池阵列的太阳位置追踪器中均有应用。

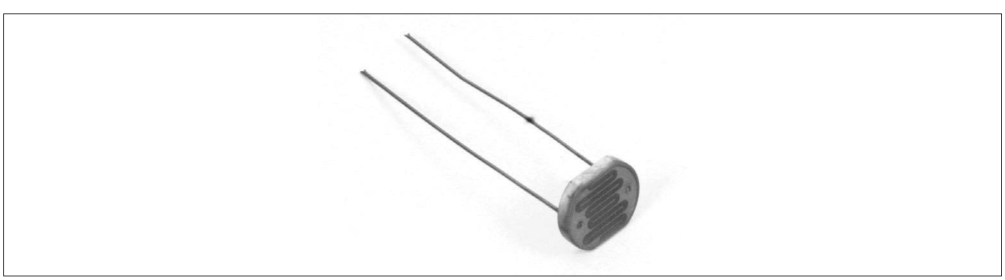


图 9-24：常见的低功耗 LDR 设备

2. 光电二极管

尽管大多数二极管对光线都有一定敏感度，但光电二极管在制造时特意增强了这一效果。顾名思义，光电二极管是人造二极管，有光线照射时就会导通。常见的光电二极管是 PIN 二极管。其中，I 代表本征半导体层，它介于二极管的 P 型半导体与 N 型半导体之间，正是这个 I 层让 PIN 二极管成为一个非常好的光线探测器。由于光电二极管的响应速度极快，因此可以把它们用在光纤数据通信链路中，也可以用作旋转机构的位置传感器。你也可以在高频无线电电路中发现 PIN 二极管的身影，它们被用作开关。关于二极管与其他固态器件的更多信息，请参考附录 D 列出的资料。

图 9-25 显示了如何把光电二极管连接到 Arduino。请注意，二极管是反向偏置的，也就是说，只有光线照射到它才能正常导通。然而这个电路只在足够亮的光源照射下才能工作，在低光照水平下则不会。

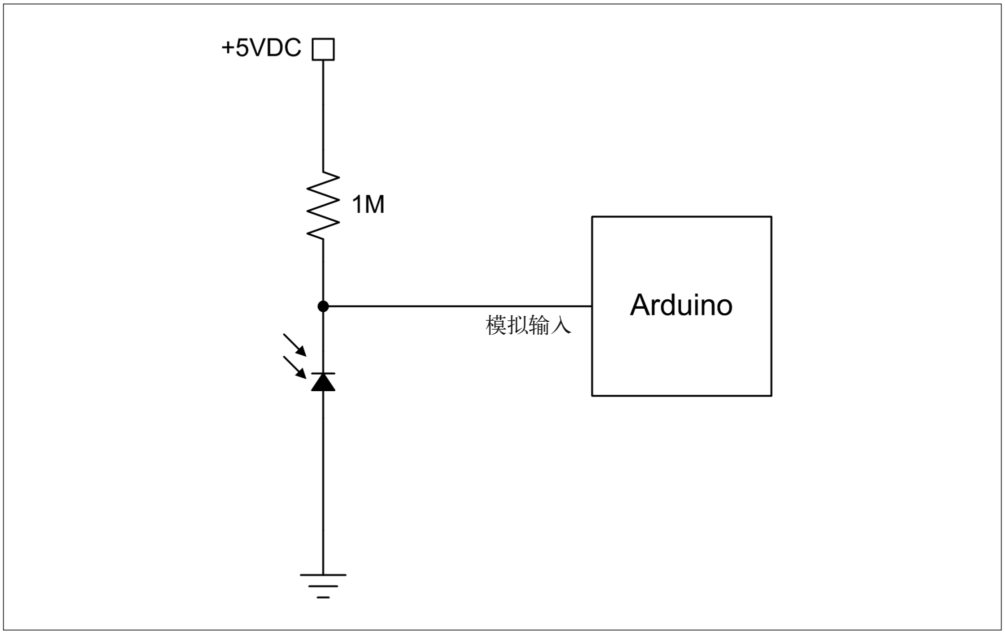


图 9-25：将光电二极管连接到 Arudino

提高光电二极管灵敏度的一个方法是使用运算放大器（op amp），如图 9-26 所示。本示例使用了 LM358 运算放大器，其最大增益约为 10 倍。二极管导通时，LM358 运算放大器放大电压的微小变化，这样 Arduino 中的 ADC 就能比较容易地检测到并进行转换。微调电位器用于设置电路增益，你可以调整它，以使其符合某种特定的应用场合。这个电路也能被轻松安装到一个小型免焊面包板模块上，或者将这些电路元件安装到 PCB 原型板上，形成一种更持久的安排方式。

你可以从电子元器件分销商那里购买到各种光电二极管（以及运算放大器），比如 Digikey（http://www.digikey.com）、Mouser（http://www.mouser.com）、Newark/Element14（http://www.newark.com）。许多过剩电子元件供应商也有光电二极管库存。

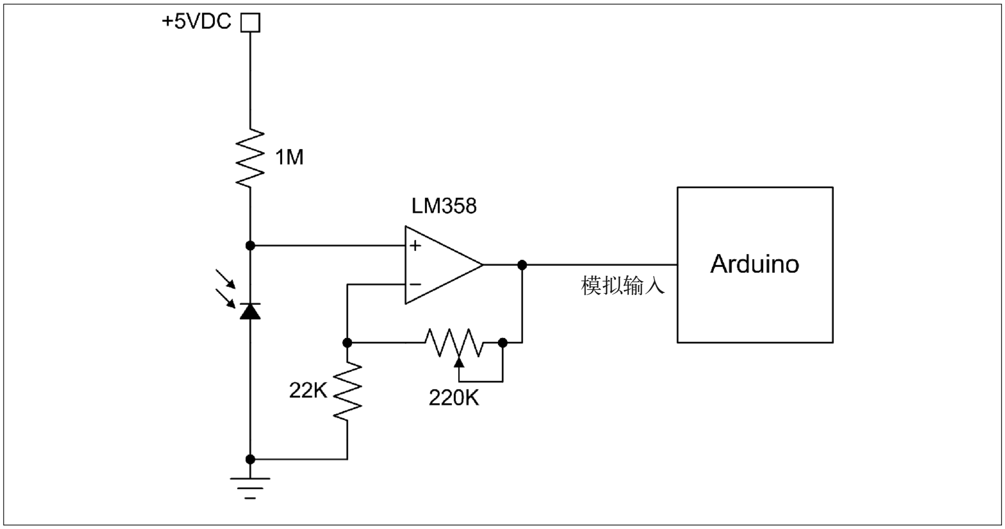


图 9-26：将带有运算放大器的光电二极管连接到 Arduino

3. 光电晶体管

顾名思义，光电晶体管能够对光线的照射产生响应，改变通过设备的电流量，正如基底输入端所做的那样。图 9-8 所示的电路与光电晶体管一起工作。有些模块——比如 KY-039——会简单地将晶体管的极脚引出到连接器引脚上（顺便说一下，你可以剪掉 LED，把该模块用作光电晶体管传感器模块）。

通过添加用于提高增益的运算放大器，你可以对基本电路进行扩展，如图 9-27 所示。几乎任何一个普通的 NPN 光电晶体管都能工作，但我喜欢 BFH310，主要因为我购买一大袋元件时获得了一个很大的优惠（随时查看各大电子元器件分销商的动态，比如 DigiKey 或 Mouser）。

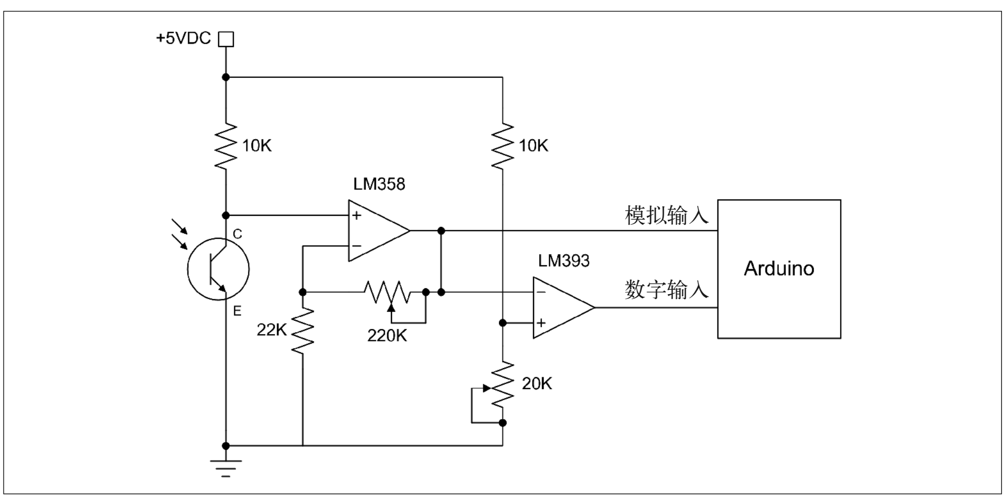


图 9-27：带有模拟与数字输出的光电晶体管电路

常见的光遮断器（比如 KY010 模块使用的那个）采用一个光电晶体管感知 LED 发出的光线。当有东西进入中间的缝隙而遮断光线时，晶体管将停止导通。光隔离器（又称为“光耦合器”）也采用一对“LED- 光电晶体管”，将信号从一个电路接合到另一个电路，而不必有直接的电气连接。你可以自己使用 LED、光电晶体管和一段黑色热缩管（套在外部，用于阻止散射光）制作耦合器。

4. PIR传感器

PIR（passive infrared，被动式红外）传感器用于测量其视野范围内红外线的数量，经常被用在安保系统中。因为它们通常可以检测出一个房间中周围环境 IR 光线的微小变化。如果检测到 IR 水平超出环境中的 IR 基准（比如有人进入房间），设备就会发出一个信号。PIR 传感器也可以测量视野范围内某个物体的大致温度。图 9-28 显示了一个常用的 PIR 模块。



图 9-28：PIR 探测器

你可以从 Banggood（http://www.banggood.com）或其他分销商那里花大约 2 美元购买到这一模块。它有 3 个接线端，分别为 +5 V、接地、输出。当传感器探测到周围环境的 IR 有变化时，就会产生高电平输出。如果将其与图 9-23 所示的声音传感器组合在一起，再使用示例 5-5 中的基本安全系统代码，就能构建一个完整的防盗报警系统。

9.4.5 磁场传感器

半导体技术取得重大进展的一个领域是对静态磁场的探测。相比较而言，探测振荡磁场（比如由线圈形成的磁场）要容易一些，因为需要的只是另一个线圈。而探测静态磁场（比如永磁体周围的磁场或地磁场）会更难。在电子设备发展之前，磁场传感器通常集成了磁铁、线圈、反射镜，以及其他元件。标准的野营罗盘是老式技术的一个例子，它可以用在徒步旅行中，帮助用户辨别方向，也可以用于探测有连续直流电通过的导线周围的磁场。但利用它收集数据则相当繁琐。如今，你可以自己制作一个磁场探测器或一个电子罗盘，而不用移动那些直接与微控制器相连的元件。

1. 霍尔效应传感器

霍尔效应传感器用于检测是否有磁场存在。一些霍尔效应传感器（比如 KY-003 中使用的

A3144）被设计为 on/off 类型的设备，它们对磁场的探测结果要么是有，要么是没有。而其他一些霍尔效应传感器（比如 SS49E）则是线性类型的，它们的模拟输出与感知到的磁场强度成正比。KY-024 模块就采用了线性霍尔效应传感器。

A3144 与 SS49E 看上去都像小的塑料晶体管，我就不在书里展示了。如果你使用 A3144 或类似的器件，可以直接将其连接到 Arduino，这在本质上与 KY-003 模块所做的一样。KY-024 模块采用了常见的比较器电路，如图 9-8 所示。

2. 磁强计传感器

磁场传感器配合 Arduino 使用的另一种形式是罗盘，如图 9-29 所示。该模块来自 Adafruit，使用一个 HMC5883L 三轴磁强计，并采用 I2C 接口。



图 9-29：磁强计罗盘模块

9.4.6 振动与敲击传感器

振动与敲击传感器通常用于检测某种可感知物体的运动。其中，可感知物体可以是简单的机械臂，它的一端有一个小物体，还排列着一些触点，这样机械臂偏转时就会与某触点接触而闭合电路。另一种形式可能要使用光传感器，当机械臂遮断光线，光传感器的输出状态就会发生改变。它也有可能是带有触点的弹簧滑体、光传感器，甚至是磁场传感器。此外，还可以使用压电传感器检测可感知物体的运动。

一种非常便宜的振动传感器采用一个密封于某种外壳的小导电体。导电体移动时，就会与外壳末端的导电套管接触而导通。KY-020 就是带有这种传感器的典型模块。

敲击传感器与振动传感器类似，但有时它被设计为只对达到一定程度的力（比如几个 g 的力，1 g= 地球平均海平面上的重力大小）做出响应。KY-031 就是一个小型的低功耗敲击传感器模块。

你可以自己使用金属球（BB 就很好）、弹簧（可能来自于圆珠笔）、一小段塑料管以及一些细针距的导线制作敲击或碰撞传感器。图 9-30 显示了如何把这些部件组装在一起。

如果需要更高精度，你可以购买工业级的敲击传感器，它们只会对特定级别的力做响应。你可以用它们进行汽车碰撞测试，也可以为易碎设备测试集装箱的撞击承受能力。但是，这类传感器的价格都不便宜。

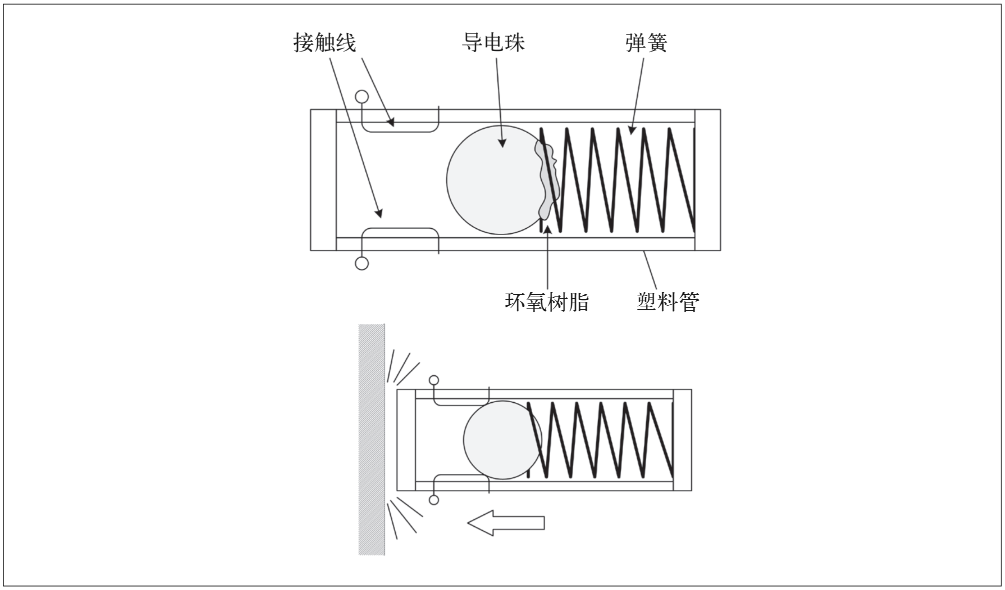


图 9-30：自制碰撞或震动传感器

9.4.7 运动传感器

感受角度变化（在原位置上以不同速率进行）的能力是在三维空间中保持物体稳定的关键。许多四轴飞行器（有时称为“无人机”）集成了某种形式的多轴运动感应器，借以模拟一个真正的机械陀螺仪或惯性测量单元（Inertial Measurement Unit，IMU）运行。这些器件在遥控飞机与直升机的爱好者当中非常受欢迎。一些富于冒险精神的人甚至把它们集成到模型火箭中，用以追踪并记录火箭在动力飞行期间的运动轨迹。

随着电子技术与制作技术的发展，这些器件的价格大大降低，这在以前是无法想象的。过去，一只固态速率陀螺仪或加速度传感器的价格大约能涨到 50 美元，而现在只需要大约 10 美元。由于成本低，再加上 IC 元件采用了非常小的表面贴装封装方式，带有细间距引脚，这使得购买整个模块比购买散件自己进行组装更有意义。当然，如果你想把这些元件用作更大部件的一部分，并且也有能力处理表面贴装元件，那么可以购买零散元件，然后根据需要进行组装。

1. 陀螺仪

在数字感知器件中，使用 gyroscope 或 gyro（陀螺仪）这个词有点不恰当。不同于真正的机械陀螺仪，这些器件更像是角速度传感器，它们用于感知围绕某个轴向的运动。从根本上说，它们并不像真正的陀螺仪那样需要一个惯性起始位置作参考。IMU 可以这样做，但大部分机电式 IMU 器件又大又笨重，内部带有 3 个（或更多）高速陀螺仪，含有精密的滚珠万向节，以及电动发动机与位置传感器。它们往往也非常昂贵。然而，借助一些巧妙的编程技巧，并使用多轴加速度传感器（稍后介绍），可以对 IMU 进行模拟。图 9-31 显示了一个三轴速率陀螺仪模块，你可以从 DealeXtreme、Banggood 或其他供应商那里购买。

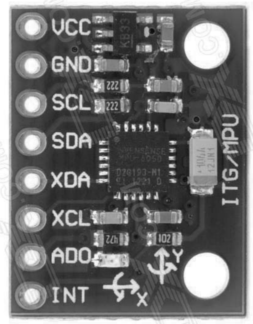


图 9-31：三轴速率陀螺仪模块

2. 加速度传感器

加速度传感器能够感知沿着某个特定线性轴的速度变化。当加速度传感器以恒定速度移动时，它不会侦测到任何变化。但发生加速或减速时，加速度传感器就会产生一个输出。三轴加速度传感器的设计使得它可以检测到速度在 x、y、z 轴上的变化。图 9-32 显示了一个便宜的加速度传感器模块，它是一款基于 MMA7361 的单轴加速度传感器，来自 DealeXtreme。你也可以从 Adafruit、SparkFun 以及其他供应商那里购买类似模块。

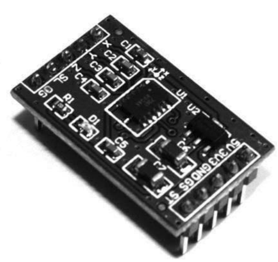


图 9-32：加速度传感器模块

9.4.8　接触与位置传感器

在各种应用场合，你总能看到接触传感器的身影，从饮料灌装机到机械加工车间中的数控机床。音乐家所用的音效箱上的那些按钮就是某种接触传感器。不管采用何种方式制造，接触传感器的工作原理都相同：它们与某部分要么有物理接触，要么没有。

顾名思义，位置传感器用于感知某个物体的位置。不同于接触传感器，位置传感器通常能够感知接近程度与角位移（角旋转的量）。一些位置传感器利用反射光，其他一些则利用声音，还有一些集成了经过特殊设计的转子与光束，用来测量角位移。另一种形式的位置传感器（称为“绝对编码器”）使用内部玻璃圆盘，带有非常细的标记，用以确定转轴的旋转精度。此处不会介绍绝对编码器。

1. 接触开关

接触开关可以简单到是一个导线“触须”，由一根弹簧及贯穿其中的导电棒组成，就像你在儿童玩具“小机械虫”中见到的那样。弹簧“触须”弯曲会使弹簧与中间的导电棒接触在一起，形成闭合电路。也就是说，它只是一个开关。图 9-33 显示的是这种接触传感器的特写。

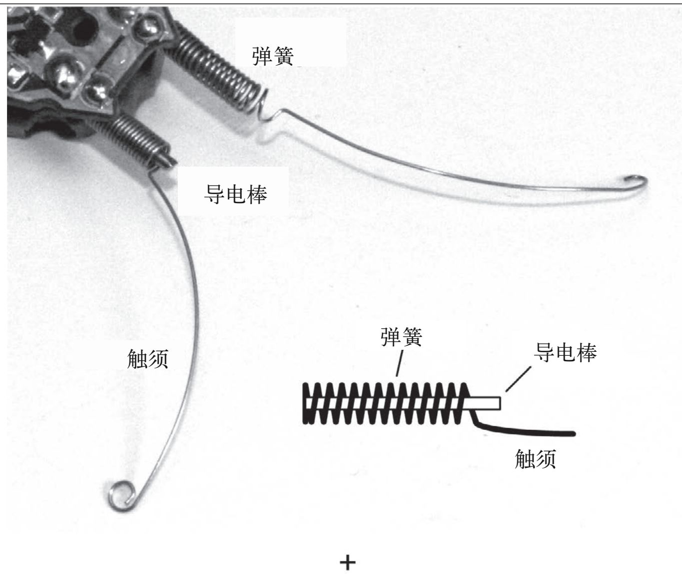


图 9-33：弹簧与触须接触传感器

另一种接触开关被称为微动开关（snap-action switches），如图 9-34 所示。该模块是 Meeeno 碰撞传感器模块，在机器人限制传感器或数控机床（CNC）等应用场合很常见这种类型的开关。

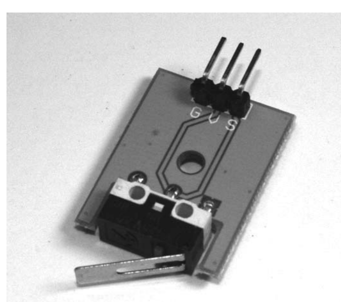


图 9-34：典型的微动开关

按钮开关也可以用作接触传感器。汽车中使用的那些老式推按开关在车门敞开时打开内部照明灯，它们是非常好的接触传感器（可能需要一定大小的力才能打开，这种力大于普通

小机器人所能传递的力）。甚至一根铜条、一颗螺丝也能用来感知物理接触，如图 9-35 所示。最重要的是，开关可以把电路闭合（或打开，视具体情况而定），并且 Arduino 能够感知到这一点，且能对此做出响应。

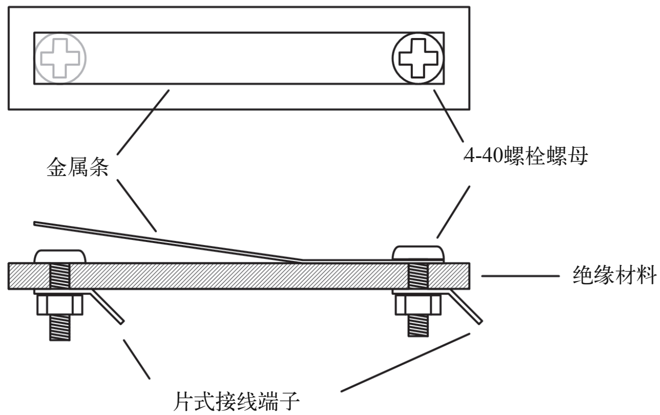


图 9-35：使用金属条制作的简单接触传感器

2. 数字旋转编码器

数字旋转编码器如图 9-36 所示，转动转轴时，它会产生脉冲或者发出一个数值。KY-040 模块也采用旋转编码器。一些旋转编码器的转轴上带有棘爪，旋转转轴时会有轻微的锁紧之感。对于不需要用户转动旋钮的应用场合，转轴可以自由、连续地转动。第 12 章给出了一段示例代码，向我们展示了如何编写用于读取旋转编码器（比如 KY-040）的软件。

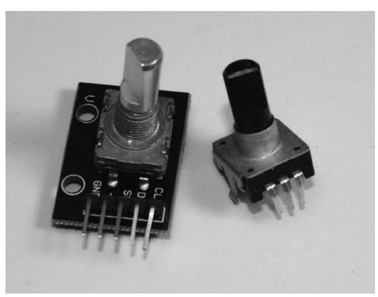


图 9-36：数字旋转编码器

老式电脑鼠标使用带有涂层的金属球——而非 LED——以及 2 个旋转编码器，它们用来探测金属球的运动。现在几乎看不到这些配件，但它们内部的确有一些有趣的器件。把它们拆开可以看到带有均匀分布凹槽的塑料轮。塑料轮转动时，从 LED 发出的光束就会被遮断。通过感知脉冲时间，鼠标中的微控制器能够产生数值，指示鼠标在一个平面上沿着 x 轴与 y 轴方向的移动距离。图 9-37 显示一款老式鼠标的内部结构。

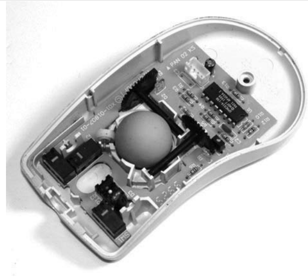


图 9-37：带有滚球与旋转编码器的老式鼠标

在这款特殊的鼠标中，LED 发射器与光电晶体管（也可能是光电二极管）是两个独立的元件。其他老式滚球鼠标采用光线遮断器元件，在表 9-7 中的 KY-010 模块上可以看到它们。光栅轮由转轴驱动，而转轴与金属球侧面接触。如果你用过滚球鼠标就会知道，有时需要取出滚球，清除转轴上的脏东西（滚球上的脏东西也要清除）。你可以拆下光栅轮与光学传感器，把它们用在其他地方。坦率地说，滚球鼠标中的光栅轮可能是最有用的零件。

3. 激光发射器与接收器

图 9-38 中的模块是一个短程激光发射器与接收器，主要用于探测障碍物，或完成其他利用光反射感知物体的任务。它也可以被用作数据传输器，但必须对光学器件进行改进，以便到达任何重要区域。



图 9-38：短程激光目标探测器

9.4.9　距离传感器

在许多机器人的应用场合中，一项关键功能是探测对象，以及测定目标对象与传感器之间的距离。距离传感器一般采用反射原理，可以是光线、声音或者无线电波（雷达）。此处只讲解光线与声音，雷达传感器往往更昂贵，并且测量的距离也更远。对于短距离感知来说，光学或声学传感器工作得很好，并且价格低廉。

1. LED物体探测器

LED 物体探测器的工作原理是，测量从 LED（或者是光线，或者是 IR）发出并被物体表面反射回来的光线。这些器件一般有一个 IR LED 与一个并排的探测器，如图 9-39 所示。它们不会测量光线从发射器发出到返回传感器的时间，因为距离很短，光线速度很快，这使得测量用时变得极其困难。如果想利用阿波罗登月时留下的一个反射镜测量地球与月球之间的距离，那么使用一个脉冲激光器、一台望远镜、一个高精度计时器将能出色地完成这个任务。而对于 Arduino，反射式传感器用来防止小机器人碰到墙体，或者让机器人沿着地面上指定的路线移动。

2. 超声波测距仪

上了年纪的人可能都会记得老式“拍立得”相机，它带有一个用于自动对焦的超声波测距仪，拍完后能立即“吐出”一张显好影的照片。通常，距离测距仪由一对压电式传感器组成，其中一个用作发射器，另一个用作接收器。发射器产生超声波短脉冲，回声会被接收器探测到。这种测距仪中，脉冲输出与回声返回之间的时间由传感器与反射体间的距离决定。超声波测距仪之所以能够正常工作，是因为声音的移动速度相对较慢，使用足够快的逻辑获得脉冲发出与回声返回的时间差并不特别困难。

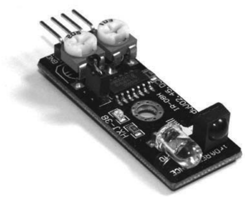


图 9-39：反射式目标探测器模块

如今，你只需要花几美元就能买到一个超声波测距仪，它可以连接到 Arduino。表 9-8 中的 20-019-100 模块就一个比较容易购买到的超声波传感器。

9.5 通信

市面上的许多模块都可以用于通信应用，从普通的 RS-232 适配器到无线通信，再到激光发射器 / 接收器模块。

9.5.1 APC220无线模块

APC220 收发器模块的工作频率为 418 MHz~455 MHz。一个完整的数字链路由两个模块与一个可选的 USB 适配器组成，其中一个模块连接到 PC，另一个模块连接到 Arduino。APC220 能以 19 200 位 /s 的速度传输数据，最大传输距离可达 1000 m。8.4.22 节介绍过多功能扩展板，其上就为 APC220 模块预留了连接点，DFRobot 的 16 × 2 LCD 扩展板（第 8 章）也是如此。图 9-40 显示的是一对 APC220 模块。

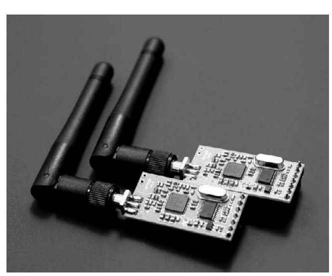


图 9-40：APC220 RF 收发器模块（图片来源：DFRobot）

9.5.2　315/433 MHz RF模块

315/433 MHz RF 模块的最大传输距离为 150 m，它们是 APC220 的廉价替代品。不足之处在于，它们不是收发一体的，一个套装由发射器与接收器两部分组成，如图 9-41 所示。使用前可以先将其频率设置为 315 MHz 或者 433 MHz。请注意，使用时需要你自己添加天线。

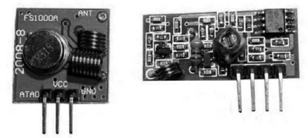


图 9-41：433 MHz 发射器与接收器

9.5.3　ESP8266收发器

如图 9-42 所示，ESP8266 收发器高度集成了 Wi-Fi 模块，支持 802.11 b/g/n 协议，使用串口与 Arduino 通信。板载 32 位 MCU 在其固件中包含 TCP/IP 网络协议栈。它处理数字链路建立与维持的低层细节，所以 Arduino 需要指定一个连接地址，或者等待某个模块与它建立连接。

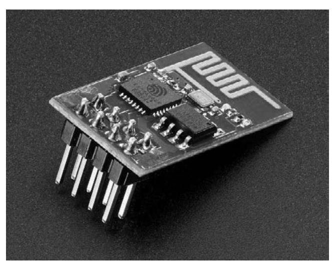


图 9-42：Wi-Fi 收发器模块

1. NRF24L01

如图 9-43 所示，NRF24L01 是一个低功耗收发器，工作在 2.4 GHz 频率之下，传输距离约为 250 m。它使用 SPI 接口与 Arduino 通信。你可以从多个渠道购买到这些模块，价格大约 3 美元。

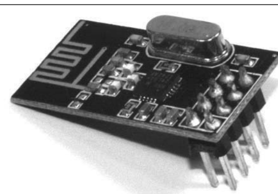


图 9-43：NRF24L01 RF 收发器

2. RS-232适配器

Arduino 开发板使用的 AVR MCU 器件内置有 UART（用 Atmel 的术语表示为 USART），但它不产生标准 RS-232 信号。你无需自己制作转换器，RS-232 适配器模块本身提供转换器、IC 与 DB-9 连接器，如图 9-44 所示。MCU 的 RxD 与 TxD 引脚直接连接到模块。

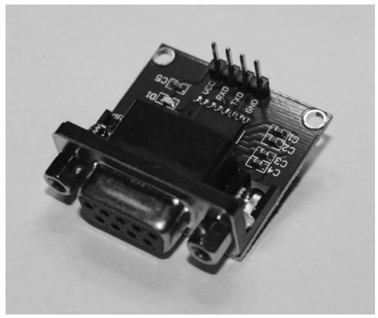


图 9-44：RS-232 适配器模块

9.6 输出设备与元件

Arduino 的输出可以是 LED、伺服电机、继电器，或者其他模块、元件，或者是对 Arduino 上 AVR 微控制器的信号或命令做响应的设备。本部分先介绍光源，接着介绍继电器、电机、舵机。当然也介绍声源设备，比如表 9-7 列出的 KY-006 脉冲响应扬声器。有关用户输入与输出元件的内容将在后面章节介绍。

9.6.1 光源

光源多种多样，从各种形状、大小的老式灯泡到现在的 LED。LED 是一种固态元件，它看上去像二极管，但有电流通过时会发出亮光。本部分将重点介绍 LED，主要是因为它们比较便宜，使用寿命长，并且不需要驱动电路。一只非常小的白炽灯会有明显的电流消耗，它们发光时会发出大量的热。话虽如此，你仍然可以使用白炽灯，只是要为外加的复杂电路做好准备，也要准备随时更换坏掉的灯泡。

LED 的种类、大小、颜色多种多样。AVR 微控制器能为 LED 提供 5 mA~10 mA 的工作电流，但通常不建议你将大量 LED 直接连接到它，也不鼓励你直接用 AVR 驱动大电流输出的 LED 模块。针对此问题，一个最好的解决方案是使用某种驱动器。在接下来的一系列图片中，你将看到不同类型的 LED，它们也可以被安装在多种 PCB 模块上，比如表 9-7 与表 9-9 列出的那些。

1. 单色LED

单色 LED 大小各异，有微小的表面贴装元件（比如 Arduino 开发板上的 D13 LED），也有巨大的闪光与照明元件。图 9-45 显示了一些不同类型的单色 LED。

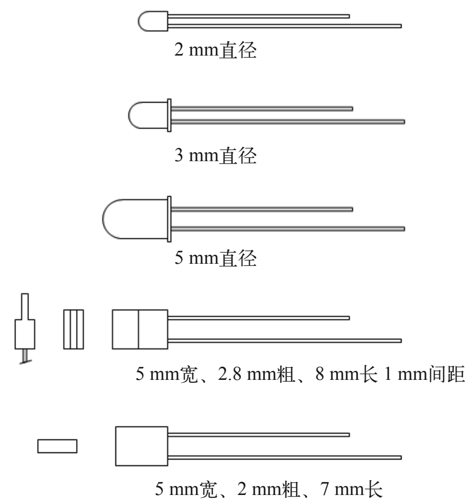


图 9-45：常见单色 LED

2. 双色LED

本质上，双色 LED 只是两个封装在一起的 LED，它看上去就像一个带有内部连接的正常 LED，如图 9-46 所示。内部两个 LED 相互反接在一起，当电流从某一端流入时，其中一个 LED 会发光；电流从另一端流入时，另一个 LED 会发光。

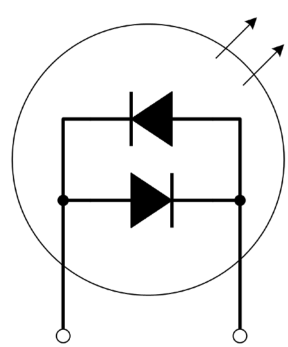


图 9-46：两接线双色 LED 内部连接

还有一种双色 LED 采用 3 根接线，其中一根共用，其他两根分别连接到各个 LED 的芯片上，这些芯片位于元件的塑料封装内部。

3. 三色（RGB）LED

三色 LED 由 3 个独立的 LED 芯片组成，它们在同一个塑料封装内。图 9-47 展示了一个表面贴装元件。大量三色 LED 能以阵列的形式安装在单个 PCB 上，多个 PCB 并排安装在一起，组成一个巨大的全彩 LED 显示器。

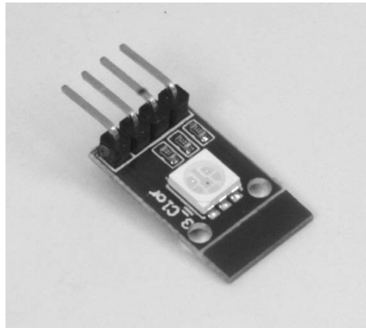


图 9-47：表面贴装式三色 LED

通过改变内部每个 LED 的输出强度，三色 LED 几乎能够模拟任何可见颜色。因为 LED 芯片（单个 LED 芯片）在物理上是独立的，颜色由某种柔光镜混合在一起——从远处看的确如此。大功率、大输出的 RGB LED 用于制造大型彩色显示屏，就像你在高楼大厦的墙体或大型体育馆中看到的那样。

4. LED矩阵

LED 矩阵用途广泛。图 9-48 显示一个 8×8 的 LED 矩阵，它可以显示字母或数字。如果把许多这样的模块并排放在一起，你就可以制作一个动态文字显示屏。

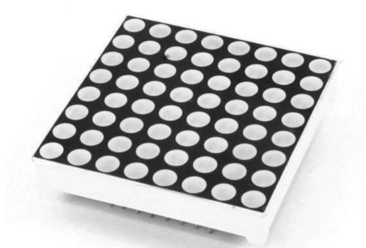


图 9-48：8 × 8 LED 矩阵

请注意，这个模块被设计成 LED 与模块的边缘之间带有一定间隔。当把多个 LED 矩阵模块并排放在一起时，相邻模块的最后一行或一列 LED 并在一起，它们之间的间隔与模块上其他 LED 灯之间的间隔是一样的。当把多个模块拼在一起制作更大的显示屏时，保持一致的间隔会让拼装显得很自然、整齐。

5. 七段LED显示器

七段 LED 显示器历史悠久。不同于单个 LED，七段 LED 显示器是 LED 技术的第一个可行的应用。到 20 世纪 70 年代晚期，七段 LED 数字显示模块（可显示字母与数字）开始应用于各种场合。图 9-49 展示了一个典型的四字显示模块。

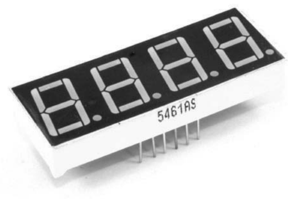


图 9-49：典型的四字七段 LED 数字管

与图 9-48 中的 LED 矩阵模块一样，该模块边缘也留有一定间隔。这样，将多个模块并排在一起时，它们能够自然地拼接在一起。使用 3 个七段 LED 数字管，你可以轻松制作出一个显示 12 位浮点数的显示器。

6. 七段LED模块

七段 LED 显示模块内置有 SPI 或 I2C 接口，如图 9-50 所示。你可以从 Tindie（见表 9-11）购买到这种特定的模块。此外还有其他带有多种显示的模块，它们可以显示为红色、绿色、黄色与蓝色。

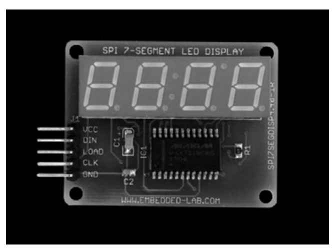


图 9-50：带有 SPI 接口的七段 LED 显示模块

7. 激光器

一些二极管光源也是激光器，比如激光笔中的那些。而其他一些激光能量非常高，可以切断塑料或木头。固态激光器本质上就是一个带有内部微调的 LED，以产生相干光。激光器 LED 输出的波长覆盖从红外线到蓝光的范围。若没有这些激光器，就不会有建筑工程中使用的激光水平仪、授课中使用的激光笔、3D 建模中使用的表面轮廓描绘仪、CD 与 DVD 播放器与录制器，以及工业中的切割工具。图 9-51 展示了一个典型的 LED 激光器模块（准确地说是 KY-008）。你可以从电子元器件分销商那里单独购买到这种激光器 LED。

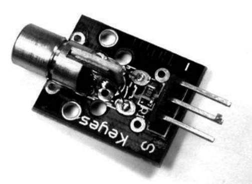


图 9-51：典型的低功耗红色 LED 激光器

9.6.2　继电器、电机与舵机

与 Arduino PCB 的 AVR 微控制器传递的电流相比，许多小型继电器能够传递更大的电流，为内部线圈提供更大电力。电机与舵机是非常有用的设备，它们工作时需要使用某种驱动器为其提供所需电流。

1. 继电器

继电器的大小、外形多种多样，既有小封装型继电器（比如 14 针的 DIP IC），也有用于控制工业设备中大电流的大型继电器。对于大部分 Arduino 应用而言，小型继电器就能完全满足实际需要。小型继电器能够控制稍大些的继电器，然后稍大点的继电器能够控制更大的继电器，以此类推。KY-019 就是一种继电器模块，它能直接连接到 Arduino。

一个简单的继电器驱动器可以是一个 2N2222 晶体管，也可以是一个专门被设计用于处理电流和反向尖峰电流的 IC。图 9-5 中的电路使用了一个 NPN 晶体管，控制一个小的 PCB 继电器。

图 9-52 显示的模块带有 4 个继电器。PCB 上也包含所需的驱动器晶体管、电阻器和二极管。使用时需要的就是一个 5 V DC 电源，用于驱动继电器中的线圈和标准逻辑信号，以便控制它们。

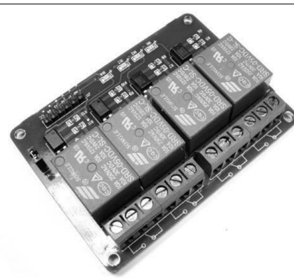


图 9-52：带有 4 个 PCB 继电器的模块

2. 舵机控制

虽然“舵机”这个术语曾用来指又大又笨重的驱动器，比如海军战舰上的火炮定位或执行模拟计算的设备，但现在通常指的是电机等小型设备。典型的小型舵机如图 9-53 所示，它们常用在遥控车、飞行器以及业余机器人中。



图 9-53：一些小型舵机

虽然有些扩展板专门用于支持这些舵机（第 8 章），但实际上，AVR 微控制器可以通过自身的 PWM 输出直接驱动它们。比如图 9-53 中的那些舵机，它们的驱动轴能够旋转 180°，其旋转角度由控制信号的脉冲宽度与频率决定。

3. DC电机控制

DC 电机通常由“H - 桥”电路进行控制，“H - 桥”的简化电路图如图 9-54 所示。“H - 桥”电路可以产生连续的直流电（DC）或 PWM 信号，依据驱动方式的不同，电机可以正向运行，也可以反向运行。

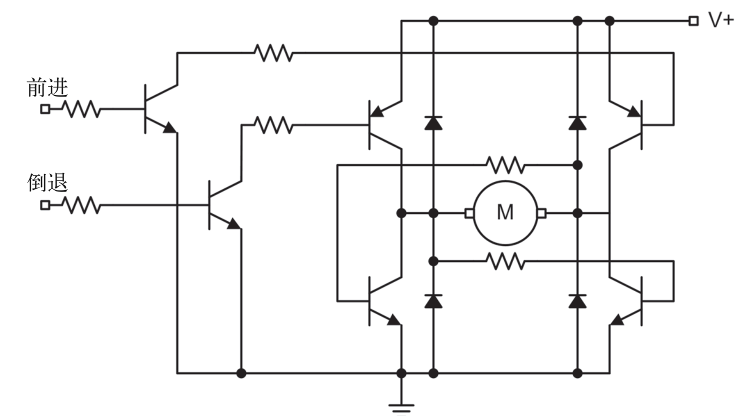


图 9-54：简化的 H 桥电路

我不建议大家自己动手搭建电机控制电路（除非你真想这样做）。扩展板上拥有控制一个 DC 电机所需要的一切，如图 9-55 所示。它也带有一个散热片，以便驱散大电流负载产生的热量。这个扩展板来自 Seeed Studio（http://bit.ly/seeed-motor-v2）。请注意，该扩展板可以控制两个 DC 电机或一个步进电机。

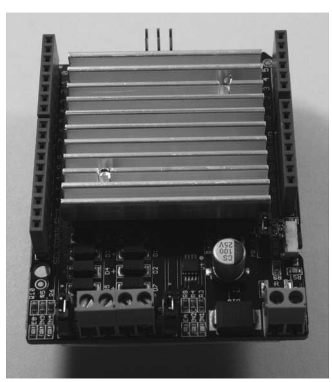


图 9-55：电机控制扩展板

4. 步进电机控制

控制步进电机要相对简单些，只要有必要的电子器件，并恰当地组装在一起产生脉冲，即可驱动电机轴旋转。当然，还需要有电流驱动它。你也可以使用 ULN2003A 等 IC 驱动步进电机工作，它提供了驱动步进电机所需的一切，但不能驱动需求大电流的步进电机。本质上，ULN2003A 只是由 8 个达林顿晶体管组成的阵列，所以 Arduino 必须处理所有电机脉冲的定时。

市面上有一些扩展板（第 8 章）可用，它们带有的电路能够控制 1~4 台步进电机，还配有专门的连接器，这使连接工作变得更加简单。与 DC 电机一样，使用现成的驱动电路比自己动手搭建要容易得多。如果你愿意花时间认真研究，那么可能购买到更便宜的驱动板。

9.6.3 模拟信号输出

模拟信号指连续变化的物理量表达的信息，比如日常生活中常见的声音，还包括那些超过人类听觉范围的信号，如无线电信号。有多种方法可以让 Arduino 产生正弦波模拟信号，但需要使用一些外部附加元件。AVR 微控制器没有在其设计中集成数模转换器（DAC），所以如果不想从 PWM 或时钟输出中得到方波，就需要使用其他设备产生信号。

1. 蜂鸣器

蜂鸣器可以很简单，启动时只发出单一音调；也可以复杂一点，产生可变音调。KY-006 与 KY-012 就是这两种声源的例子。

2. DAC模块

一种让 Arduino 拥有数模转换能力的方式是使用 DAC 模块，如图 9-56 所示。这个模块来自 Adafruit，它基于 MCP4725 IC，是一个带有 I2C 接口的单通道 12 位的 DAC。

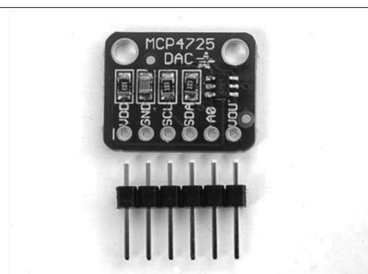


图 9-56：基于 MCP4725 的 DAC 模块

需要连续变化的信号——比如其他电路或设备的控制电压——时，DAC 模块很有用。DAC 也能产生波形，比如斜波、正弦波。

与 AVR 微控制器一起使用的 I2C 在标准模式下的通信速率为 100 kB/s，不可能达到更新速率，不过可以使用更快的 I2C 接口类型。但是，DAC 仍然能产生相当好的低频正弦波。不足之处是，AVR 只能不断更新 DAC 产生波形，此外无能为力。

3. 波形发生器

虽然使用 Arduino 与 DAC 模块可以直接产生低频波形，但要想产生超过 1 kHz 的高质量波形，必须使用外部电路。直接数字频率合成（DDS）模块就是这样一种设备，如图 9-57 所示。

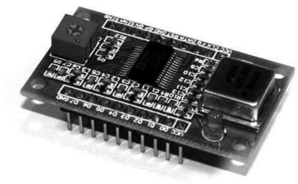


图 9-57：基于 AD9850 的 DDS 模块

AD9850 IC 既能产生方波，也能产生正弦波，频率范围为 1 Hz~40 MHz。你可以从美国模拟器件公司（Analog Devices）网站（http://bit.ly/ad9850-data）下载 AD9850 的数据手册。AD9850 使用自己的独特接口，借助 8 位并行或串行接口进行控制。AD9850 的 Arduino 库也很容易找到。

9.7 用户输入

有时需要直接与 Arduino 项目进行交互，主要手段有按钮、旋钮、键盘、摇杆等。除了本章介绍的模块外，你也可以自己购买电子元件，并根据自身需要进行组装。

9.7.1 键盘

“键盘”（keypad）这个术语通常指一系列呈一定方式排列的开关，这些开关带有键帽，常见排列方式为 3 × 3、3 × 4 或 4 × 4 网格。它也可以用来指薄膜式键盘（membrane keypad），薄膜式键盘是在 PCB 上按一定方式排列的一系列薄膜开关。按键上常常标有字母与数字，如图 9-58 所示。键盘并不仅局限于小的矩形阵列，你可以看到各种类型与布局。事实上，电脑键盘只是一个大的“键盘”。

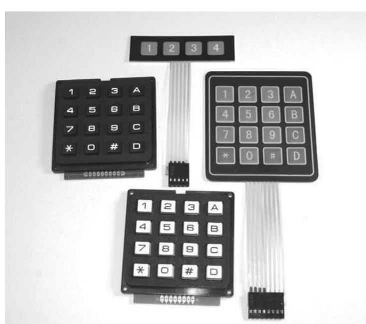


图 9-58：各种各样的键盘

9.7.2 摇杆

摇杆通常有两种类型，一种是连续模拟式的，另一种是离散数字式。图 9-59 展示了一个连续模拟式的摇杆，它带有两个电位器，分别连接到 x 轴与 y 轴。电位器的读数表明摇杆的移动距离以及当前位置。

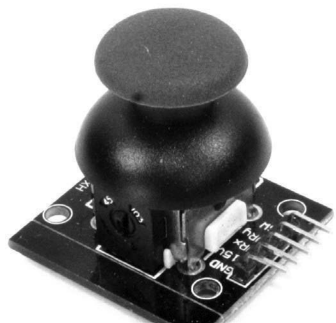


图 9-59：模拟摇杆模块

当摇杆沿 x 轴或 y 轴方向（或者两者同时）移动到最大位置时，离散型数字摇杆会采用小开关或其他某种类型的探测器进行检测。这些类型的摇杆一般用在早期的低成本游戏与个人电脑中。它们是“全有或全无”（all-or-nothing）设备，但制造成本低，也不怕灰尘与磨损（影响模拟摇杆正常工作）。许多 LCD 扩展板集成了离散摇杆。

9.7.3 电位器与旋转编码器

电位器是一种阻值可变的电阻器，常用来控制输入。电位器或可调电阻可以用在调光器模块中，也可以调节音量、控制测试仪器的输入以及各种模拟电路的输入控制。TinkerKit T000140 与 T000150 模块都是电位器。

旋转编码器如图 9-36 所示，它可以用作一个用户输入设备。可变电压通常必须转换为数字，这样 AVR 微控制器才能使用它。旋转编码器不产生可变电压而产生数字输出，可以直接使用。

9.8 用户输出

通过向用户显示信息，你可以设计出能与用户进行交互的作品。这些信息可以是 LED 上显示的简单状态，也可以是 LCD 或 TFT 显示器上显示的复杂信息或图像。无论何种形式，输出设备都会对用户输入的命令及时做出响应与反馈。

市面上有各种各样的显示设备可以与 Arduino 一起使用。第 8 章介绍过 LCD 显示器扩展板，它采用了相同的显示器件，但提供了更方便的使用方式。然而，某些场合可能不方便使用扩展板，此时使用 LCD 显示元件并按所需方式进行安装可能是更好的选择。

第 8 章介绍了多种显示器扩展板，它们都能与 Arduino 一起使用。请务必阅读。除非你真的需要使用带有裸露引脚的显示器，否则选用扩展板是使用显示器的更简便的方式。

9.8.1 文本显示器

相对而言，文本显示器更常见，也更便宜。它一般能显示 1~4 行文本，每行可以显示 8、16、20、24、32、40 个字符。当然，随着字符显示密度的增加，价格也会相应上涨。例如，一个 1 行 8 字符的显示器大约 2 美元，而一个 4 行 40 字符的显示器价格则为 18 美元左右。

1. ERM1601SBS-2

ERM1601SBS-2 LCD 显示模块是一个 16×1 显示器，白字蓝底，如图 9-60 所示。它采用一个 HD44780 或 KS066 控制芯片与 LED 背光照明，售价约为 3 美元。此外，你也可以选用其他类似的产品，比如黑字黄绿底的、黑字白底的。



图 9-60：ERM1601SBS-2 显示模块

2. ST7066（HD44780）

这是一款 16×2 行 LCD 显示器，带有简单的并行接口，采用 HD44780 或 ST7066 控制器。在 Adafruit、SparkFun、SainSmart 与其他厂商的 LCD 显示器扩展板上，你可以看到一样的元器件。图 9-61 显示的就是这样一款显示器，售价大约为 10 美元。



图 9-61：采用 HD44780 或 ST7066 控制器的 16 × 2 显示器

你可能已经注意到，图 9-60 中的 ERM1601SBS-2 与图 9-61 中的显示模块看上去很像，因为它们都使用了相同的 IC 进行驱动 LCD。唯一的区别是，一个是单行显示，另一个是双行显示。

你可以从多种渠道购买其他类型的 LCD 显示器，既包括显示非图形的，也包括显示图形的。有时，你甚至能以很低的价格购买到新的过剩显示器，如果它们采用了标准的控制器芯片，那么通常都能很容易地被集成到 Arduino 项目中。问题是，这些剩余显示器中常常内建有特定于某个用途的元器件，所以购买后会发现，显示器是针对于微波炉或草坪洒水系统的，其实你并不需要（或者你可能正需要）。

9.8.2 图形显示器

市面上有许多图形显示器可用，它们大都采用 LCD、TFT、OLED 技术，支持单色和彩色格式。TFT 与 OLED 看上去比简单的点式寻址的 LCD 好得多，但这是要付出代价的。

1. ERC240128SBS-1

图 9-62 展示的 ERC240128SBS-1 显示器是一个 240 × 128 点式寻址 LCD 显示器，带有一个 8 位的并行接口。你可以从 BuyDisplay 网站（http://www.buydisplay.com）购买。

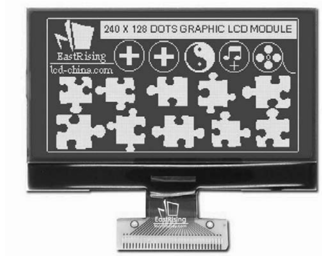


图 9-62：240 × 128 彩色 TFT 显示器

2. ST7735R

另一种显示器是来自 Adafruit 的 128 × 160 TFT 显示模块，如图 9-63 所示。这款显示器的显示大小为 1.8 in（4.6 cm，对角线），拥有 18 位色彩选择能力，支持 262 144 种不同色调。它采用一个带有 SPI 接口的 ST7735R 控制器，PCB 背面还带有一个 microSD 卡槽。

9.9 支持功能

大多数模块都提供输入或输出功能，但拥有非 I/O 支持功能的模块并不多，这样的模块通常被归入时钟与定时器类别。

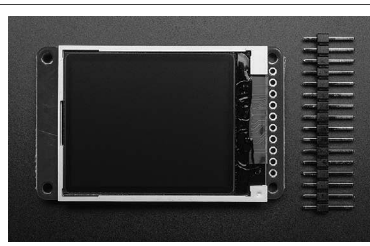


图 9-63：1.8 in 的彩色 TFT 显示模块

9.9.1 时钟

市面上有多种实时时钟（RTC）IC 可用，比如 DS1302、DS1307、DS3231 和 PCF8563。本质上，它们做的事都是一样的——记录时间与日期。其中有些拥有板载 EEPROM，有些则没有；有些使用非标准接口，有些使用 SPI，有些使用 I2C。除了接口不同之外，它们最主要的不同表现在时间的精度、温度敏感程度、耗电量以及生产成本上。

四种常见的 RTC 模块分别基于 DS1302、DS1307、DS3231、PCF8563 集成电路。这些模块通常带有纽扣电池槽，方便安装 CR2032 或 LIR2032 等纽扣电池。独立测试表明 DS3231 拥有最佳的长期稳定性，但其他 RTC 也是完全可用的。带有外部晶体的模块会受到温度变化的影响，长期运行时，精度会有一定程度的下降。

1. DS1302 RTC模块

DS1302 RTC IC（图 9-64）采用了一个非标准的串行接口。它不是 SPI，但由时钟控制。一根线用于传输数据，一根线用作时钟信号，另一根用作芯片使能（CE）线。这就是 Maxim（达拉斯半导体）所说的三线接口。图 9-64 展示了一个典型的 DS1302 模块。有关 DS1302 的更多信息，请访问 Maxim Integrated 官网，阅读相关技术文档（http://bit.ly/maxim-ds1302）。

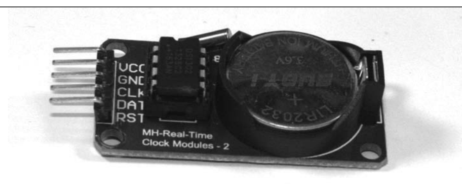


图 9-64：DS1302 RTC 模块

2. DS1307 RTC模块

DS1307 RTC 是一个 I2C 器件，在代码或引脚上不兼容 DS1302，但最终结果是一样的。阅读 Maxim 提供的数据手册（http://bit.ly/maxim-ds1307）可以获得更多相关细节。图 9-65 展示了一个 DS1307 RTC 模块，它来自 Tronixlabs（http://tronixlabs.com）。

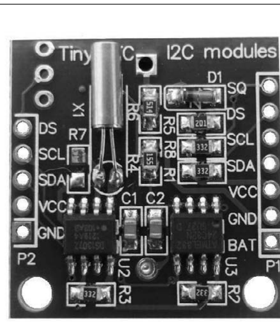


图 9-65：DS1307 RTC 模块

3. DS3231 RTC模块

类似于 DS1307，DS3231 RTC 采用一个 I2C 接口。它与 DS1307 是代码兼容的，二者的主要不同在于精度。DS3231 采用了一个内部晶体，这使其对外部温度变化不太敏感。图 9-66 展示了一个典型的 DS3231 RTC 模块。

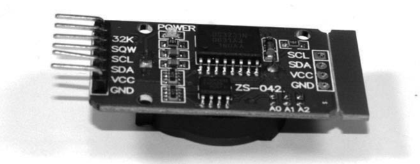


图 9-66：DS3231 RTC 模块

4. 采用PCF8563的RTC模块

PCF8563 是一个带有 I2C 接口的 RTC IC。它是 NXP（http://www.nxp.com）公司的产品，其内部寄存器完全不同于 Maxim 的 DS1307 或 DS3231。图 9-67 展示了一个典型的基于 PCF8563 的模块。

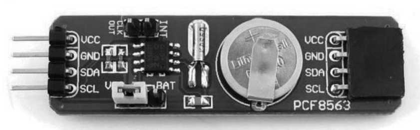


图 9-67：PCF8563 RTC 模块

9.9.2 定时器

尽管 AVR MCU 已经内置了看门狗定时器，但仍有一些模块专门提供这一功能。外接的看门狗（更准确地说是可复位的倒计时器）有一种可能的用途，比如来自外部设备的定时器重置信号比来自内部 AVR 的信号更有意义时。假设有一个旋转机构，它安装有一个磁性传感器，该传感器发射脉冲以控制转轴旋转。如果脉冲用于重置板外的看门狗定时器，那么当旋转机构故障而停止转动时，就会被检测到。通过使用中断或查询倒计时器的状态，MCU 能够检测故障条件（fault condition），并采用适当的措施。

许多不知名的板外（outboard）定时器模块采用 555 定时器。每当出现重置脉冲时，就要 MOSFET 对定时电容（timing capacitor）放电，重置定时器。其他一些来自亚洲销售商的定时器模块会使用一点黑色环氧树脂，隐藏 PCB 上实际执行计时的元件。我一般不会买这样的模块，因为要想了解隐藏在环氧树脂底下的部分，一般都要先破坏 IC 才行。

如果你对外部倒计时器模块感兴趣，建议阅读 Howard Berlin 所写的 *The 555 Timer Applications Sourcebook*（附录 D）。最早的 1979 年版本已经绝版，但在 Amazon 上仍然能找到一些副本。你也可以购买到更新版本的，但内容略有不同。澳大利亚的 Freetronics 公司（http://www.freetronics.com.au）销售一款基于 555 定时器的看门狗定时器模块，价格不贵。

9.10 连接

过去几年里，模块与扩展板的互连方法上开始逐渐显现出一种趋势：模块式连接器正在取代扩展板与模块上连接器的引脚与插口。这常常被称作“系统”，包括一系列模块与某种形式的接口扩展板，它们都使用相同类型的连接器与引脚（电压、信号、地）。

TinkerKit 模块（表 9-9）只是模块连接系统的一个例子。另一个例子是 Grove 系列模块，及其连接的的接口扩展板（来自 Seeed Studio）。其他扩展板拥有 3 个或 4 个脚的连接器，以便使用预制电缆连接相互匹配的连接器，比如图 8-21 中的无源 patch 扩展板套件。

9.10.1 使用裸露跳线

比如你打算把一个模块永久连接到 Arduino（它可能要被嵌入某个地方进行长期工作），并且不想花时间自己制作连接器。你当然可以使用跳线进行连接，这不会有什么问题，但你要采取一些简单的步骤以保证物理连接的可靠性。

跳线中使用的压接插座（引脚）端子与模块式连接器中使用的端子基本一样，不同之处在于，跳线只有一个端子，而连接器有两个或更多。连接器中压接端子越多，就会越健壮、耐用。这是由多个插座端子增加的机械摩擦引起的，这些插座端子都在同一个连接器外壳中工作。

单根跳线容易扭动与弯曲，它没有由单个连接器外壳中一群端子提供的机械抓握能力。提高连接可靠性的一个方法是使用一小滴硅橡胶（silicon rubber），将跨接连接器（jumper connector）固定到模块引脚。这可能不像模块式连接器那样优雅、牢固，但只要模块不在有剧烈震动的环境（比如遥控车或工厂中的机床）下工作，使用硅橡胶会有不错的固定效果。

请不要使用太多硅橡胶，因为某些时候你可能想移除它并替换那个模块。在此情形之下，你可以使用锋利的剃刀刀片切掉软硅橡胶，而不会损坏跳线（当然，必须足够小心才行）。

9.10.2 模块连接系统

一般而言，带有连接器的扩展板（不管是 TinkerKit 使用的开放式架构类型，还是 Grove 和其他厂商使用的闭壳类型）也可以与单独的跳线或压接插口头一起使用，就像带有裸露引脚的扩展板与模块一样。预制连接器使得用户可以更轻松地连接模块，而不用担心引脚如何连接，只要模块被设计为连接到一个带有同类型连接器的扩展板上即可。这就是 TinkerKit 与 Grove 模块采用的方案。你可以从 Seeed Studio 网站（http://www.seeedstudio.com）获取更多关于 Grove 模块的信息。

其他系统——比如 TinyCircuits 推出的 TinyDuino 模块——采用小型表面贴装式的多引脚连接器。图 9-68 展示了这些模块的几个例子。严格来说，这些模块并不属于本书介绍的任何一种，它们向我们展示的只是用于处理互连问题的众多方法之一。TinyCircuits 也生产带有相同类型连接器的传感器模块，并提供相应的延伸电缆。它们非常小，第一次看到它们时，首先浮现在我脑海里的是“模型火箭”这个词。在 TinyCircuits 网站（https://www.tiny-circuits.com）可以找到更多相关信息。

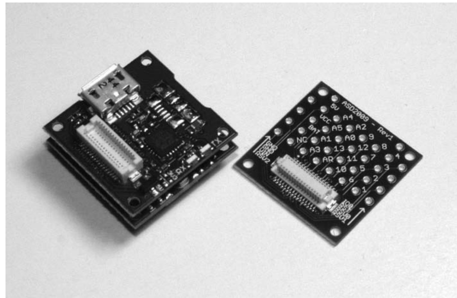


图 9-68：TinyCircuits 生产的模块样本

使用针对特定互连系统的模块与扩展板时，连接器确实是个大问题。不同制造商之间没有统一标准，这就无法保证一个供应商的模块能够插到其他供应商售卖的接口扩展板上。解决该问题的方法之一是，制作基本接口扩展板与一系列配套模块，就像 TinkerKit 所做的那样。考察特定模块时，请务必关注它使用的连接方法。你还要准备购买附加电缆，以便把针对其他特定连接系统（connector system）设计的元件连接到一些原型模块（prototyping modules）上。当然，你也可以自己动手制作连接线缆。

9.10.3 自己动手制作连接器

连接器不仅易于使用，也比跳线更牢固、可靠，但没有连接器并非都是坏事。请看图 8-3，它是一个来自 SainSmart 的传感器接口扩展板，其 I/O 引脚一行行整齐地排列着，间距为 0.1 in（2.54 mm）。图 9-69 显示的连接器头带有相同的间距与压接端子插孔，能够与引脚匹配，形成牢固的连接。

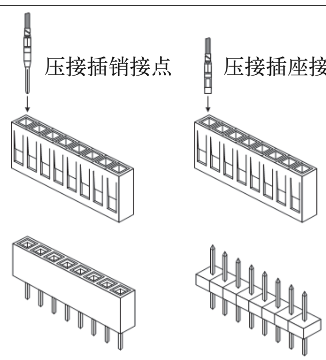


图 9-69：压接端子的引脚与插口头

连接器外壳与压接端子有多种类型。连接器外壳一般能够容纳一个或多个端子。有时，你可能会遇到末端带有单位（single-position）塑料外壳的跳线，而不是常用的热收缩管绝缘材料。我更喜欢塑料外壳，尽管这种跳线会稍微贵一些。

如果你只想连接一个或两个模块，并且不想为跳线的松动与脱落而担心，那么插口接头（socket header）是一个值得考虑的替代方案。不便之处在于，你需要额外花些钱购买压接工具，以及一把好的剥线钳。图 9-70 显示的是一把压接钳，图 9-71 展示了使用方法。



图 9-70：标准的端子压接钳

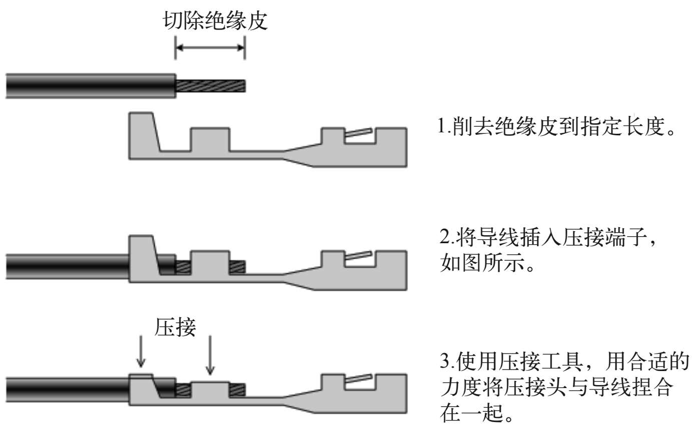


图 9-71：如何压接端子与导线

图 9-72 显示了一个三位（three-position）的插口端子接头，它连接到一个温湿度传感器模块（KEYES KY-015）。这种结构布局方式允许将传感器安装到任何需要的地方，或连接到一个接口扩展板。在连接器外壳与模块 PCB 之间可以使用少量透明硅胶（clear silicon），以保证连接器不会轻易从模块松动。

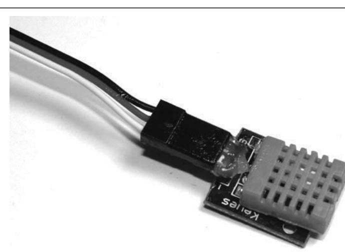


图 9-72：使用三位插口接头连接传感器模块

9.10.4 选择连接方法

对于连接方法的权衡最终可归结为，使用模块式连接器与预制电缆（可靠、牢固，但需要使用配套元件），还是自己动手进行连接；使用带有压接插口端子的跳线，还是自己动手为特定模块制作插口接头。选择哪一条路取决于你想花多少精力将模块连接到 Arduino 或扩展板，以及想付出多少时间研究特定的接线方案。

你可能已经注意到，我讲解模块连接方法时并未提到“焊接”。它始终是一个可选项，但应该作为最后的选择，除非你真想把某个模块永久连接到某个项目应用中。将一个模块直接焊接到某根导线或 PCB 上可以形成牢固的连接，但并不容易拆卸，而且外观也不好看。同时，这也意味着你少了一个可以用到其他项目中的模块。

话虽如此，你可能已经注意到，图 9-12 中，继电器与温度传感器模块被直接焊接到 Arduino 温控器的原型扩展板上。原因在于，那个模块以后任何时候都不会被取出而停止服务，它将一直在电暖器不太好的环境中运行，容易受到热扰动与风扇震动的影响。所以，我决定采用焊接的方法。虽然这样做可能会让我后悔，但到目前为止，它的工作一切正常。一旦遇到类似的情形，你就需要自己拿主意，决定是否采用“焊接”这个方法。

9.11 供应商资源

表 9-11 只列出了销售 Arduino 可兼容元件与模块的部分供应商，这些公司只是我碰巧知道的，并且和它们中的大部分进行过交易。此外还有许多其他我不知道的销售商，如果在互联网上仔细查查，你可能会发现它们。最后提醒你，不要忘了去 eBay 逛逛。

表9-11：电子元器件供应商

分销商/供应商	网址	分销商/供应商	网址
Adafruit	www.adafruit.com	Mouser Electronics	www.mouser.com
Amazon	www.amazon.com	RobotShop	www.robotshop.com
CuteDigi	store.cutedigi.com	SainSmart	www.sainsmart.com
DealeXtreme (DX)	www.dx.com	DFRobot	www.dfrobot.com
Seeed Studio	www.seeedstudio.com	Elecfreaks	www.elecfreaks.com
SparkFun	www.sparkfun.com	Elechouse	www.elechouse.com
Tindie	www.tindie.com	Freetronics	www.freetronics.com.au
Tinkersphere	tinkersphere.com	iMall	imall.itead.cc
Tronixlabs	tronixlabs.com	ITEAD Studio	store.iteadstudio.com
Trossen Robotics	www.trossenrobotics.com	KEYES	en.keyes-robot.com

9.12 小结

本章介绍了各种不同的模块与元件，它们与 Arduino（或者任何现代微控制器）一起使用能够实现输入输出功能。如前所述，你可以在一个扩展板上找到许多这里介绍过的功能，但有时可能没有合适的扩展板可用。通过使用 Arduino Nano、一些传感器、输入元件、某

类输出，你可以把它们组合在一起，以满足自身特定需求。

借助 Web 浏览器访问互联网，你可以轻松找到数百种不同的 I/O 元件，它们都可以连接到 Arduino，你甚至也可以找到裸露的 AVR IC。从电子元器件分销商（比如 Digi-Key、Mouser、Newark/Element14）到生产厂商（像 Adafruit、SparkFun、SainSmart、CuteDigi），你有很多渠道可以购买这些电子元器件。此外，还有一些分销商专门经营低价的剩余电子元器件（surplus components）。

为 Arduino 项目购买传感器时，你要考虑如下几点：

(1) 该设备使用简单接口（离散数字、SPI、I2C 等）吗？
(2) 该设备带有技术资料吗（或者容易获取吗）？
(3) 有 Arduino 软件库可用吗？

其中，第 3 条是否重要取决于你的编程水平。就个人而言，我更看重前两条。主要因为，相比于逆向分析一个从 eBay 购买的很酷的复杂接口，我有更重要的事情要做。对我来说，购买不那么酷的“小玩意”，然后获取所需要的技术资料并让它跑起来，这样更有意义。最主要的是，购买那些能够完成指定工作的电子元件，并且价格要在可承受的预算之内。

第10章

自己动手制作元件

通常，越经常使用 Arduino 元器件（特别是 AVR 微控制器），你就越有可能体会到它们的灵活性与通用性。你能想到的任何应用似乎都能找到可用的传感器与扩展板，并且新扩展板也在不断涌现。即便如此，仍然会有一些应用找不到现成的扩展板。有时，你可能在网上花费数小时搜索带有特定功能的扩展板，但最终一无所获，这才意识到原来根本就不存在这样的扩展板。在此情形之下，你大体有 3 种选择：一是放弃当前的方法，尝试寻找另一种解决方法；二是找人为你制作（通常需要付费）；三是自己动手设计并制作 PCB。本章将介绍两个项目，阐述步骤过程中会涉及自己动手制作扩展板与 Arduino 软件兼容元件的内容。

第一个项目是制作一个扩展板，如图 10-1 所示，它专为特定场合的应用设计，我称它为 GreenShield。这个扩展板基于传统的扩展板外形规格，采用表面贴装元件布局，包括电位器、继电器、LED。

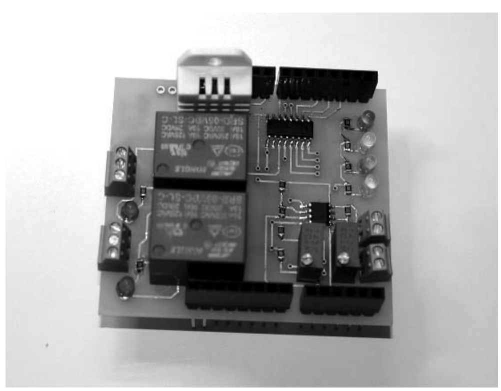


图 10-1：GreenShield

与 Arduino 接在一起应用于园艺与农业时，GreenShield 能够充当独立的监测器与控制器。该扩展板也能用在自动气象站、风暴预警监控器、温控器（我们将在第 12 章介绍使用现成模块与传感器制作的可编程温控器）中。

本章后半部分将介绍 Switchinator，它是一个基于 AVR ATmega328 的设备。虽然未搭载 ArduinoBootLoader 固件，但仍然可以借助 Arduino IDE 与 ICSP 可编程设备进行编程。Switchinator PCB 如图 10-2 所示。

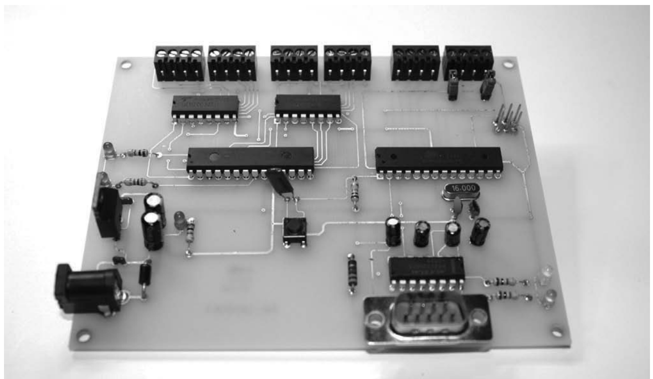


图 10-2：Switchinator

通过 Switchinator 可以远程控制高达 14DC 的设备，比如继电器或大电流 LED，也可以驱动 3 台单极步进电机，或者使用外部固态继电器控制 AC 负载。它采用 RS-232 接口，不需要经过 USB 连接到主机 PC。

Switchinator 将 Arduino 必需的所有元件集成到一个我们自己设计的开发板上。无需担心插口接头的大小，也不需要考虑 PCB 扩展板的布线。对整体大小与形状的唯一约束是我们要在设计中实现的那些目标。

只要稍微有些耐心与规划，你就能轻松制作自己的 PCB。虽然外形与 Arduino 不同，但同样易于编程。最重要的是，它能准确实现你的设计目标，物理外形也恰好符合你的需要。

制作 PCB 并不那么困难，但需要有一些 PCB 设计与电子电路的知识。有很多便宜或免费的软件工具可以处理 PCB 布局，制作一个电路板其实相当容易。学习本章中的项目时，还需要你具备一点焊接技术，尤其是安装表面贴装元件时。如果你经历过这些，那么就离成功不远了。如果没有，那么学习如何制作电路图，如何使用 PCB、电烙铁，以及如何选择合适的元件，将是一段快乐又宝贵的经验。

制作扩展板 PCB 时，我们将使用 Eagle 电路设计与 PCB 布局工具；而对于 Arduino 兼容的 PCB，我们将使用 Fritzing 工具进行制作。这两个工具都很常用，且功能强大，最重要的是，它们都是免费的（Fritzing 是免费的，入门级的 Eage 软件免费，但功能有一定限制）。

自己制作扩展板或 Arduino 兼容板时，准备一个笔记本是非常有用的。即便它只是三孔活页夹中一些打印或复印的页面，可如果以后你需要查看类似项目的信息，你也将非常庆幸自己做了笔记。也许有人会问：为什么不把它们保存到 PC 的磁盘中呢？因为磁盘驱动器发生故障且未备份时，记录在其中的内容会丢失。有时，也会出现磁盘存储内容太多而丢失的情况（我有时会遇到这种情况，尽管我不太想承认）。而且，当你在测试、制作、部署方面获得一些经验后，如果之前做过笔记，你将能很容易地返回相关部分，把这些经验、心得加注其中。请记得，红色批注笔可不是只有高中英语老师才能用的。

本章也提供了一个资源列表，涉及软件、配件与 PCB 制作。讲解相关内容时，我假设你在电子方面已经拥有一些经验，或者至少愿意多花精力学习一些相关的基础知识。请一定要看看附录 A 中的工具概述，以及附录 D 中的阅读建议。最后，不要忘记研读一些著名网站的文章，比如 Hackaday（http://hackaday.com）、Makezine（http://makezine.com）、Adafruit（http://www.adafruit.com）、SparkFun（http://www.sparkfun.com）、Instructables（http://www.instructables.com）。另外，你也可以在 YouTube 上搜到相关教程视频。许多前人走过类似的道路，做过类似的尝试，他们把自己的经验记录下来并与大家分享，让我们每个人都受益。

请注意，本书讲解的重点是 Arduino 硬件以及相关模块、传感器、元器件，书中出现的示例代码仅用于突显要讲解的重点内容，它们可能不是完整的可运行代码。相关示例与项目的完整代码可以从 GitHub 上进行下载（https://www.github.com/ardnut）。

10.1　准备工作

与任何值得投入大量时间的努力一样，做计划是极其重要的。这不仅适用于复杂软件的开发与实现、建造房屋、设计一级方程式赛车、组织北极探险，也适用于电子项目。正如古语所说："做不好打算等于打算失败。"

每一个重要的项目都可以被分解为一系列步骤。通常，制作一个电子器件需要 7 个基本步骤。

(1) 定义
(2) 规划
(3) 设计
(4) 原型
(5) 测试
(6) 制造
(7) 验收检测

有些项目可能需要更少步骤，有些可能需要更多，这取决于你具体制作什么。每个步骤的详细介绍如下。

定义

以正式的工程术语来说，这被称为“需求定义阶段”，更准确地说，是“功能需求定义阶段”。对于比较简单的器件，使用一份简要的描述说明该项目要实现的最终结果，以及如何使用它就够了。而对于复杂的元器件，则可能需要一份更加详细的需求定义描述，比如要制作一个在 CubeSat 上与 Arduino 一起使用的扩展板。但无论复杂性如何，把它写下来能够帮助记录过程步骤，也有助于检查之前未发觉的遗漏或错误，避免陷入无法轻易做出实质性改变的境地。最后，写下项目定义也是个好主意，这样就可以用它测试原型或成品设备。如果功能性需求定义未能清晰描述设备应有的功能，测试人员也无法使用这种描述对设备进行测试，那么显然，它就不是一份合格的需求描述，未能准确定义需要的设备与系统。虽然写功能需求听上去挺无趣的，但它的确是工程中一个很重要的方面。你都不知道要去哪，又怎么会知道什么时候到呢？一个好的需求（任何类型）应该包含 4 个基本特征：（1）应该保持自身以及总体设计的一致性（2）应该条理清晰、连贯一致，并且具有意义（3）必须简明扼要，不过于啰嗦（4）必须是可验证的。这意味着，在功能需求中应该使用“该设备能够把 250 ml 烧杯中的 100 ml 水在 5 min 内加热到 100℃”这类语句，而不能使用类似于“该设备能够把水加热”的描述（加热到何种程度？需要多长时间？多少水量？）。

规划

有时，做规划与定义需求同时进行，当然前提是要制作的器件相对简单且易于理解。但无论怎样，在规划阶段都需要为设计阶段收集必要的信息，还要尽可能多获取聚集在一处的重要信息，比如元件的数据手册、部件来源，以及对于必需设计与软件工具的确认等。该阶段的目标是，准备好设计阶段需要的一切，以便能做出正确的判断，知道有什么可用、要多长时间，以及如何使用。

在规划阶段，你也可以对后续的每个步骤耗费的时间做些有根据的猜测，包括设计、原型、测试、制造、验收检测。我之所以说是“有根据的猜测”是因为，收集了尽可能多的信息之后，你对这些应该心中有数。但这仍然是猜测，毕竟你不是巫师，没有窥视未来的“水晶球”。有时会发生意想不到的事，有时完成项目的某些方面耗费的时间可能要比实际预测的时间长。这反映出现实世界中事物是如何发展的，也解释了从事项目管理的人为什么会因为两三个因素的出现而增加对时间的评估。毕竟，过多估计时间和提早完成项目要比过少估计所需时间与延误交付好得多。

对于建立切实可行的时间表以及作为一种衡量进度的方式，使用规划工具是非常方便的。我最喜欢的快速制作进度表的工具是时间线图表，其中最常用的就是甘特图。它们可以是使用项目管理软件或图 10-3 所示的简单图表创建的复杂事务。

对于许多小项目，并不需要多么花哨的图表，增加复杂度意味着要做许多额外工作。做规划时要注意如下几点：（1）所有必要的任务都要记入规划之中（2）从时间与资源角度看，计划是切实可行的（3）计划有明确的目标与终结点。对于简单的时间表，另一件需要注意的事情是，一些任务在前一个任务完成之前就开始了，时间表并不指示关键路径的依赖关系。我发现，对于仅涉及一个或几个人的小项目，这种重叠的日程安排能够更真实地反映事件发生的实际情况。

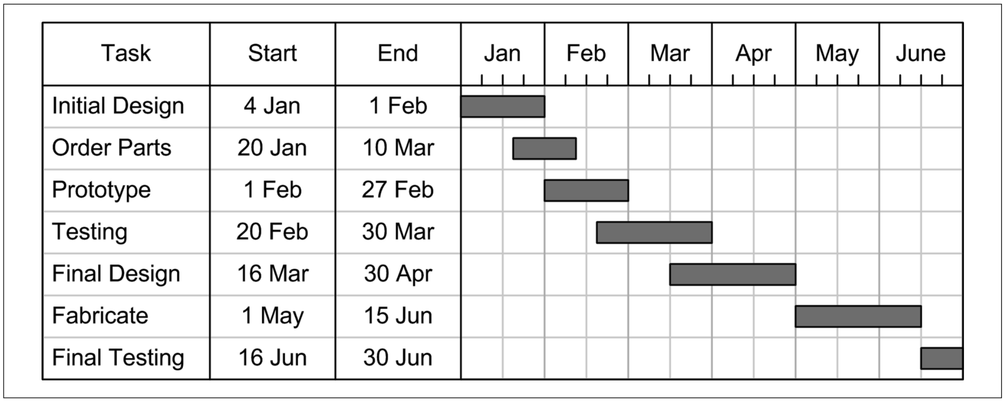


图 10-3：时间表示例

设计

对于硬件项目，在设计阶段要画出电路图，还要设计将采用的物理外形。借助手头的项目定义与规划信息，你应该能很容易地搞清这个阶段需要做什么。在设计阶段，除了大多数人能够想到的电路定义之外，还包含其他一些重要活动，比如选择具有合适外形与大小的元件、评估元件与环境因素（湿度、震动、温度）的电气规格，甚至还要考虑潜在的射频干扰（radio frequency interference）问题。

设计一个新器件时，总是需要做大量选择、决定。有时，选择这个方法而不选择另一个的原因是基于成本、是否有可用的零部件以及物料的考虑。“要实现的功能”是另一个需要考虑的因素，比如输入控制需要执行多个功能的情形。有时，选择可能基于审美因素，特别是在不论走哪条路都不会产生额外成本时。最后，在某些情况下，做选择时通常会选择自己熟悉的东西，而不会选择自己不了解的事物。尽管这并非总是做选择的最佳理由，但这种情形的确经常发生。

不管哪种项目（硬件、软件、结构等），在设计阶段，通常需要迭代进行。那些希望在设计的初次尝试中就获得成功的想法是不切实际的，除非这个设计对你根本不重要（即便如此，我也不希望你这样做）。事实上，设计与做原型协同工作，可以发现潜在问题，制定可行的解决方案，进一步改进设计。这在工程中是常见做法，虽然有时很让人恼火，但设计中的迭代是改进设计必不可少的组成部分。

改了又改

在第 11 章信号发生器的开发过程中，设计经历了几次重大修改。原计划想通过 RS-232 接口与并行数字模式输出功能实现对发生器所有功能的远程控制，但最终结果表明，不使用 I/O 扩展板与 RS-232 扩展板的情形下，Arduino 没有足够的 I/O 引脚来实现这一设想。虽然可以使用其他方法（比如使用旋转编码器）解决一些问题，但是每一种替换方案都有自己的缺陷。设备自身仍然需要按钮来控制一些功能，从可用的显示空间看，LCD 变得非常狭小。

所以，我想让设备尽量保持简单，不会向硬件与软件加入太多复杂性。最终设计不会在软件中使用中断（它们不需要使用），也不会让 LCD 上挤满各种艰涩难懂的数据值。只需要两个主要的 PCB 组件，一个是 Arduino 开发板，另一个是 DDS 模块。作为一个信号发生器，最终设计的设备将提供许多功能，但不支持任意模式输出或远程主机控制功能。

原型

对于一些简单的设备，比如一个基本的 I/O 扩展板（不带有源电路），或许真的没有必要为它们制作原型（有的根本做不了）。在其他情况下，原型可以用于对设计进行验证，保证设计的设备能够根据项目开始时给出的定义进行运转。比如，有一个设备在同一块 PCB 上集成了 AVR 处理器、LCD 显示器、蓝牙收发器、多轴加速度传感器、电子罗盘，那么直接为它设计 PCB 是不可取的。有些设备刚开始工作一切正常，直到为它制作 PCB 并且为之付款后，一些从未想到的小问题才显现出来。先制作并测试原型不仅能为你省去日后的许多麻烦，还能节省一笔不小的开支。

对于原型中发现的问题，可以再次反馈到设计并做相应改进。在一种极端情形下，原型甚至可以证明最初的设计是错误的，你可能需要重新设计。虽然制作原型可能有些烦，但并不是一场灾难（其实比你想得更常见）。制作 100 个电路板之后，突然发现设计中有重大缺陷，这才是灾难。制作原型、进行测试、修正设计，能够防止这样的灾难。

测试

测试的根本目的在于搞清两个问题：设备能够根据预期设定正常工作吗？设备能否像预想的那样安全、可靠地工作？最初的项目定义就是衡量标准，判断设备是否具备期望的功能，以及安全性与可靠性是否达到预期。可以是对基本功能的简单测试，也可以是十分复杂的测试。如果你设计的设备不会被发射到太空（或海底），或者并不用来控制那些昂贵的设备（一旦发生故障会造成巨大经济损失或造成严重伤害），那么对原型进行基本的功能测试应该足够了。

“正确”“安全”“可靠”这些词之间是有区别的：不能仅仅因为设备能够以正确的方式工作，就认为它是安全的（安全也意味着运行没有带来不可接受的风险）。有些设备能够安全工作，但未必正确、可靠。如果一个设备不启动，那么它可以被认为是安全的、可靠的（此处的“可靠”指什么都不做），但这肯定是不正确的。而一个设备是正确且可靠的，并不意味着它是安全的。比如手持式电动圆锯可以正确、可靠地切割废旧家具，但它也可能正确、可靠地切断人手。一个内部短路的电子设备在通电时将会“可靠地”发出烟雾（甚至冒出火焰），但整个操作既不安全，也不正确。

制造

对原型进行测试并确定它能够按照功能要求正确运行后，接下来进入制造阶段。本阶段包含制作 PCB，然后将各个组成部件安装到 PCB 上。此外，该阶段还可能涉及将预制模块、相关电缆与导线集成到一个外壳或更大的系统中。制造过程中，基于某些原因，可能发生延误问题。比如供应商已经没有某个电路元件的存货，或者签订好组装合同后，装配厂发生了一些问题或出现超额预订。定制的部件也可能由许多原因造成交付延迟。

幸运的是，如果你只制作一个或几个部件，并且独自完成所有制造工作，那么能避免许多潜在的问题。并且，给自己多点时间也无妨，比如去硬件商店购买一盒 3/8 in 6-32 机械螺丝，如果那家商店没有存货，你就得再花一些时间去另一家商店看看。当然，如果在规划与设计阶段就关注装配制作中需要的元器件，那么你应该能够较为轻松地获取自己需要的所有元件、PCB、螺母、螺栓、螺钉、垫片、连接器、导线、托架、黏合剂。

验收检测

最后一步是验收检测，也是最终检测，它是设备在投入使用之前要做的最后一件事。前面已经对设备原型的基本功能做了测试，这一阶段要对最终产品进行测试。这绝对不是对前面测试工作的重复，因为我们很容易把错误元件安装到 PCB 上，或者把元件装反了。此外，有时 PCB 上的电路与在免焊面包板上搭建的原型电路在行为上有所不同，所以仔细测试装配好的设备总是好的。

通常，验收检测分为两阶段进行。第一个阶段验证设备或系统是否拥有与原型一致的行为方式，采用的方法是应用与原型相同的测试，检验测试结果是否发生变化。在软件工程中，这一过程称为“回归测试”。第二个阶段主要验证设备能否在最终的配置环境（带有实际 I/O）中正常工作。这正是将此阶段称为验收检测的原因，验收检测主要回答如下问题：该设备或子系统满足实际应用的需要吗？

10.2 制作扩展板

设计一个 Arduino 扩展板时，基本上有 3 种主要布局可以考虑，分别是基线布局、扩展布局、Mega 引脚布局。Duemilanove、Uno R2 以及其他老式开发板采用原始或基线（亦称 R2）布局，这是一种标准的扩展板布局方式，但并不意味着设计扩展板时不能采用 Uno 与 Leonardo 扩展板所用的扩展布局方式（也称为 R3）。此外，也可以将扩展板设计为能够使用 ArduinoMega 系列开发板上的所有引脚，扩展板的轮廓不必非得和 Arduino 一样。一些扩展板（比如带有继电器或大散热器的扩展板）的物理外形必须能够适应其上的电子元件，而不必与要连接的 Arduino 保持一致。

一种忽略大小限制的新方法是，制作一个大的 PCB，其上安排一系列引脚，以便将 Arduino 翻过来安装到 PCB 上。这听上去有些奇怪，请参考图 10-4。这款开发板来自 Roland SRM-20 桌面型 CNC 数控铣床，更多内容请参考 Nadya Peek 的 infosyncratic.nl 博客（http://bit.ly/open-hardware-footbath）。

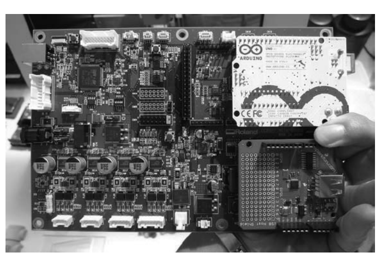


图 10-4：倒装在一个大 PCB 上的 Arduino（图片来自 Nadya Peek）

如果你正在设计一款用于商业销售的扩展板，那么可能会采用基线布局作为模板。因为它不仅能与 Uno、Leonardo 开发板很好地配合，还可以与 Mega 系列 PCB 一起使用。第 4 章介绍了各种类型的开发板，并给出了它们的大小与引脚布局信息。

Nano、Mini、Micro 这类小型 Arduino 开发板很独特，它们 PCB 的底部拥有所有 I/O 引脚，有点像大号的 IC。将扩展板添加到这些开发板需要使用一个适配器（如图 10-5 所示），它把信号引出到引脚插口，以便连接扩展板 PCB。

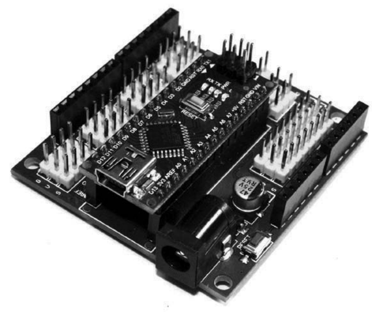


图 10-5：Arduino Nano 接口适配器 PCB

在图 10-5 中，你可能已经注意到，引脚插槽也被添加到 PCB 上。这多半是一次尝试，借此搞清将 Nano 物理连接到普通扩展板时需要什么。只要发挥一点聪明才智，任何人都可以把一个扩展板插到开发板上，但 Nano 插在插口上会显得太高。对此，一个解决方案是，为扩展板引脚使用扩展，这些扩展只是经过扩展的引脚插口，引脚长度缩短了一些。一种更激进的方案是拆掉 Nano 的现有引脚，然后用更短的引脚代替。我推荐大家使用引脚扩展方案。本章内容不会讲解如何制作这种适配器。

实际上，使用带有扩展板的 Nano、Mini 或 Micro Arduino 通常并无太大意义（尽管第 12 章中有这样的例子，但它针对的是特定的应用场景）。而把这些小的 PCB 当作大 IC 看待，并把它们作为组件用在大 PCB 上的确是有意义的。

10.2.1　物理考虑

请注意，Duemilanove PCB 上有两个表面贴装电容（其他开发板可能也如此），它们可能会与一些扩展板上的电源和模拟连接器的额外引脚接触，这些扩展板一般专为 R3 扩展基线布局而设计。Uno 的一些兼容板使用的元件也可能与扩展板的引脚发生接触。

另一个要考虑的是，Duemilanove 的 B 型 USB 接口以及 Uno 开发板和 PCB 发生短路的潜在可能性。这些开发板上的 DC 电源插座也可能对扩展板 PCB 产生干扰。尽管它是塑料的，不会引发短路，但会影响扩展板的安装。图 10-6 描述了 Duemilanove 与以太网扩展板安装在一起的情形。

基于这些原因，要么改变扩展板的长度，使其不影响底下的 Arduino；要么对扩展板 PCB 的元件布局做精心安排，把发生碰撞的区域留空。关于 PCB 大小与扩展板堆叠的更多内

容，请参考接下来的部分、第 4 章以及第 8 章。

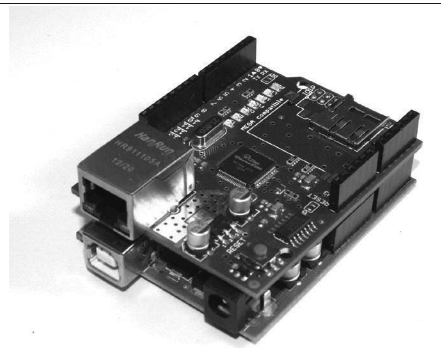


图 10-6：带有扩展板的 Duemilanove

设计扩展板时，你可能也要考虑位于扩展板之下的 Arduino 有哪些功能不能再被访问，还有什么能够安装，什么不能安装。这包括重置按钮、表面贴装 LED 以及 ICSP 引脚组。有些扩展板通过简单复制 Arduino 开发板的引脚输出解决这一问题，其他一些开发板（比如大多数 LCD 扩展板）可能不会把 Arduino 引脚复制到开发板 PCB 的正面，但有时会提供重置按钮。对于 LCD 而言，这样做是有意义的，因为它总是位于所有扩展板的最顶层。

10.2.2　堆叠扩展板

Arudino 外形的优势之一是，它可以堆叠扩展板。在 Arduino 开发板（比如 Uno）上，你可以先堆叠一个 SD 存储卡扩展板，再叠加一个输入 / 输出扩展板，最后在最顶层再叠加一个 LCD 或 TFT 显示器扩展板。瞧，现在你就有了一个基本的数据记录装置。

在一个扩展板之上可以安装什么总是一件要考虑的事情，可以是另一个扩展板，也可以是某种类型的传感器模块。为扩展板挑选组件时，考虑各种部件的高度是明智之举。如果某个扩展板上所用的部件太高，导致其他扩展板不能物理叠加至其上且不会产生任何干扰，那么该扩展板必须总是位于堆叠的最顶层。

堆叠扩展板的方法大体上有两种：一种是交错式的插口与引脚头，另一种是扩展的引脚插口头。此处的“交错”是指上部连接器（插口头）与底部引脚（引脚头）之间发生一定偏移，上部数字 I/O 与电源 / 模拟引脚被以同样的幅度移到同一边。

图 10-7 中，在 Duemilanove 开发板上，你可以看到一个模块化的 I/O 扩展板。有几点需要注意：首先，I/O 扩展板采用交错式的连接器方案，所以与底部的 Arduino 开发板并不垂直对齐。如果要将其安装到某种外壳并用在有震动的环境中（可能无法使用螺母与螺栓保护扩展板），你可能需要考虑这一点。其次，扩展板并没有把 ICSP 引脚引出，因此也就失去了相应功能（这可能是一个大问题，也可能不是）。最后，位于扩展板边缘的连接器的引脚可能与 Arduino 的电源与 USB 连接器发生接触，为此，有必要使用某种类型的绝缘垫片。

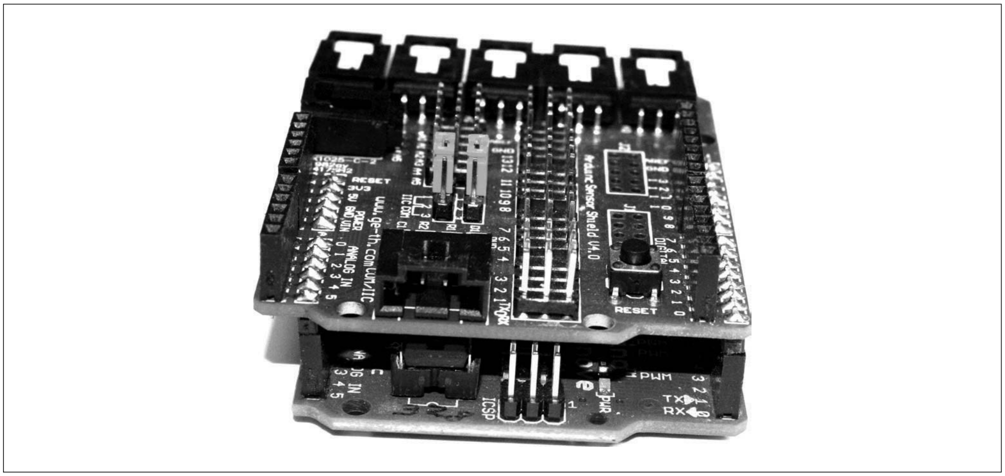


图 10-7：交错式扩展板示例

扩展引脚连接器插口是 0.1 中心连接器的常见变形，它拥有一条可以穿过 PCB 的引脚。该引脚很长，足够与底部的开发板形成牢固的连接，连接器插座允许另一个扩展板安装到其上，并与其下的扩展板保持一致。这些连接器插口出现在许多扩展板中，挑选扩展板时，你应该找找它们。

设计扩展板时，需要重点考虑 Arduino 引脚的用法。这是除了堆叠之外需要考虑的方面。扩展板所用的引脚决定着还有什么可以与处于堆叠中的那个扩展板一起使用。避免独自占用 SPI 与 I2C 引脚意味着其他扩展板也可以在堆叠中使用，包括 SD、microSD 闪存、I/O 扩展、蓝牙、ZigBee、以太网、GSM 扩展板。换言之，不要对 SPI 与 I2C 引脚做修改，除非扩展板设计真的需要你这样做。

有时，你可能会遇到一些扩展板，别人觉得很不错，但实际工作起来却不尽如人意。I/O 扩展板常常带有多组连接器引脚，如果另一个扩展板置于其上，这些引脚将无法使用。有些扩展板带有的焊盘会对 Arduino PCB 的 ICSP 引脚产生干扰。诸如此类的问题并不少见。令人遗憾的是，我们总是无法事先知道扩展板是否有问题。有时，搞清某个特定扩展板有无物理安装问题的唯一途径就是——购买它后亲自试试。

10.2.3　电气考虑

如果你要制作的扩展板只包含无源元件（比如连接器、开关、电阻器），那么板子的供电可能不会是什么问题。然而，如果它包含 LED 或有源电路，就应该考虑它需要多少电力以及从哪里获取。即使是 LED 这些简单的小电流元件，如果数量足够多，也可能导致 AVR 处理器过载并造成一定损害。

一般而言，如果一个扩展板拥有一个或一个以上的继电器或连接器，并且它们连接的每个设备耗电都在几毫安以上，那么就应该考虑使用某种驱动电路或 IC。当然，也可以使用单独电源为扩展板供电，但这样在 Arduino 与扩展板之间传递信号可能会很棘手。

如果扩展板自带 DC 电源，那么可能也需要考虑接地线。标准 Arduino 上有 3 个接地插口，其中两个位于带有模拟输入的一侧，另一个位于数字 I/O 一侧。如果只把其中一个接地插口用作扩展板的信号接地参考，则可以避免感应噪声、接地回路等潜在问题。尽管这些问题极少出现，但仍然有可能发生，特别是那些集成了高增益运算放大器或高频电路的扩展板。请记住，采用良好的设计能够帮助我们避免以后可能发生的各种奇怪且难以诊断的问题。

10.3　制作GreenShield扩展板

本部分我们将自己动手制作一个扩展板，详细介绍设计与制作 Arduino 扩展板过程中涉及的各个步骤。我们要制作的扩展板是一个湿度、文档、环境光、土壤湿度监测器，主要用在温室中，该扩展板带有合适的外壳，一些太阳能电池组件、某种无线收发器，可以被放入田地中监控芜菁（或其他东西）。世界上的某些地区（包括美国西部地区）水资源稀少，监视土壤含水量、温度、空气湿度将有助于减少灌溉次数与用水量，同时保证作物长势良好。农场主坐在家里就可以通过智能手机、平板电脑或桌面 PC 掌握田地的情况。

我之所以将扩展板称为 GreenShield，原因显而易见。设计 GreenShield 时，我们把定义阶段与规划阶段合二为一。从物理结构看，GreenShield 扩展板非常简单，硬件设计的主要挑战在于，如何将几个较大的元件（两个继电器与一个 DHT22）安排到一个小的 PCB 上。

10.3.1　目标

本项目的目标在于创建一个扩展板，将其用作自主远程监控器，用于感知温度、湿度、土壤含水量、环境光照度。根据预定义的传感器输入限制，该扩展板将控制两个继电器。

继电器可以控制水阀、风扇或者一些灯光，以弥补阴天的光线不足。扩展板带有 6 个 LED，其中两个表示湿度，两个代表土壤水分，剩下两个分别用于指示两个继电器的工作状态。

软件支持命令响应协议（command-response protocol），内部维护着一个自动中继功能表格，分别映射到传感器的输入与限制。另外，主控计算机可以根据需要直接获取传感器读入，并对继电器进行控制。

10.3.2　定义与规划

GreenShield 扩展板将集成一个 DHT22 温湿度传感器、一个光敏电阻（LDR）传感器（检测环境光，参考第 9 章）、一个基于导电性的土壤水分探头。两个继电器将自动或在命令下控制外部设备或电路。

传感器输入：

- 温度
- 相对湿度
- 土壤水分（相对）
- 环境光照度

控制与状态输出：

- 2 个控制继电器、软件功能定义、10 A 控制能力
- 4 个 LED 指示土壤水分与湿度限制
- 2 个 LED 指示继电器状态

电气接口：

- 针对于水分探针输入的双位接线端子
- 用于 LDR 连接的双位接线端子
- 每个继电器一个三位接线端子（NC、C、NO）
- Arduino 开发板提供的 +5 V DC

控制接口：

- 命令响应协议，控制主机驱动
- 按需获取传感器读数
- 由控制主机控制继电器

所有电子元件都会被设置到一个标准的扩展板上，相应大小已经在第 4 章讲解过。除了接线端子、继电器、DHT22 温湿度传感器之外，其余大部分元件都是表面贴装类型。

10.3.3 设计

GreenShield 扩展板不带有显示器，不支持用户控制。换言之，GreenShield 与 Arduino 组成自动控制系统，其中 GreenShield 充当自主远程传感器，Arduino 充当控制器。GreenShield 可以连接到其他计算机系统（主机系统），以便获取工作参数，返回传感器数据以及对继电器进行超驰控制。

此处所说的“自主控制”（Autonomous）是指，满足特定条件（比如湿度、土壤水分、照度）时，GreenShield 将自动操控继电器。软件将从主计算机接收命令，设置各种阈值水平，以及实现对继电器运行的超驰控制。它将对命令产生响应，包括当前温度、湿度、照度、继电器状态。控制主机与 GreenShield Arduino 之间的所有交互作用都是命令响应事务。

GreenShield 软件将完全使用 Arduino IDE 开发。主机计算机不仅用于编译、上传代码，当代码在 Arduino 上运行时，还用作测试终端接口。最好使用 Python 等语言自己编写接口程序，或者在 Windows 环境中使用类似 TeraTerm（https://ttssh2.osdn.jp/index.html.en）的终端模拟器。它包括一个很棒的脚本功能，我向你强烈推荐。

Eagle 电路图与 PCB 工具

在本项目中，我将使用 Eagle 电路图与 PCB 工具。如果你尚未安装，请从 http://www.cadsoftusa.com/download-eagle 下载并安装。大部分主要 Linux 发行版中可用的 Eagle 版本都比较老。从 CadSoft 网站下载的免费版本在功能上有如下限制：

- 可用开发板区域限制为 100 mm × 80 mm（4 in × 3.2 in）
- 只有两个信号层可用（顶部与底部）
- 电路图编辑器只能创建两个工作表

标准 Arduino 大约为 69 mm × 53 mm，因此，使用 Eagle 免费版本设计标准或扩展型板子都没有问题。但由于受到大小限制，它不能为 Arduino Mega 类型 PCB 的扩展板设计扩展板。设计大部分扩展板时，“只有两个信号层可用”通常不是什么问题。但对于那些要处理视频或 RF 信号的扩展板，可能还需要更多层以用作接地与电源层。

SparkFun 网站上有一些清晰简明的教程，讲解如何安装与运行 Eagle 软件。你可以在 http://bit.ly/sparkfun-eagle 和 http://bit.ly/sparkfun-using-eagle 页面找到它们。

此外，从 SparkFun 的 GitHub 仓库（https://github.com/sparkfun/SparkFun-Eagle-Libraries）也可以获取 Eagle PCB 库组件。我发现，SparkFun 不能在 Kubuntu 12.04（最新可用版本为 Eagle 5.12）自带的 Eagle 软件版本中正常工作，但可以正常安装并运行来自 CadSoft 的最新版本的 Eagle（版本 7.4.0）。我采用了一种偷懒的方法，在已经安装有 Eagle5.12 的系统中继续安装 Eagle 7.4.0，把 /usr/bin 中旧版本的 Eagle 可执行文件复制到 eagle.old，并在 /usr/bin 中创建符号链接，使其指向位于 /opt/eagle-7.4.0/bin 的新版本。

1. 设计功能

首先要考虑的是 PCB 扩展板的物理设计。图 10-8 中的框图给出了扩展板要具备的功能。

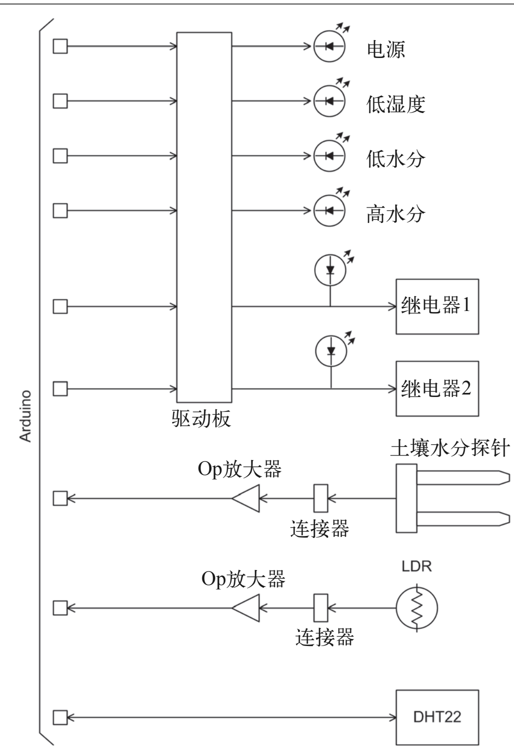


图 10-8：GreenShield 框图

请注意，图 10-8 中的硬件功能并不直接交互。传感器、LED、继电器只是对 Arduino 基本 I/O 功能的扩展。

湿度 / 温度传感器安装于 PCB 上，如果需要，光电管（光敏电阻）与土壤水分探针可以位于板外。微型接线端子用于连接传感器，因此并不需要进行焊接或压接。

GreenShield 用于堆叠扩展板的最后一层（顶层），因为继电器、温度 / 湿度传感器很高，其他扩展板不能叠加其上。

2. 硬件

从图 10-9 中的电路图可以看出，GreenSield 扩展板的电路并不复杂。ULN2003A 驱动 LED 状态指示灯与两个继电器。一个双运算放大器用于缓存来自土壤水分传感器与 LDR 的电平信号，并用作 AVR ADC 的输入。

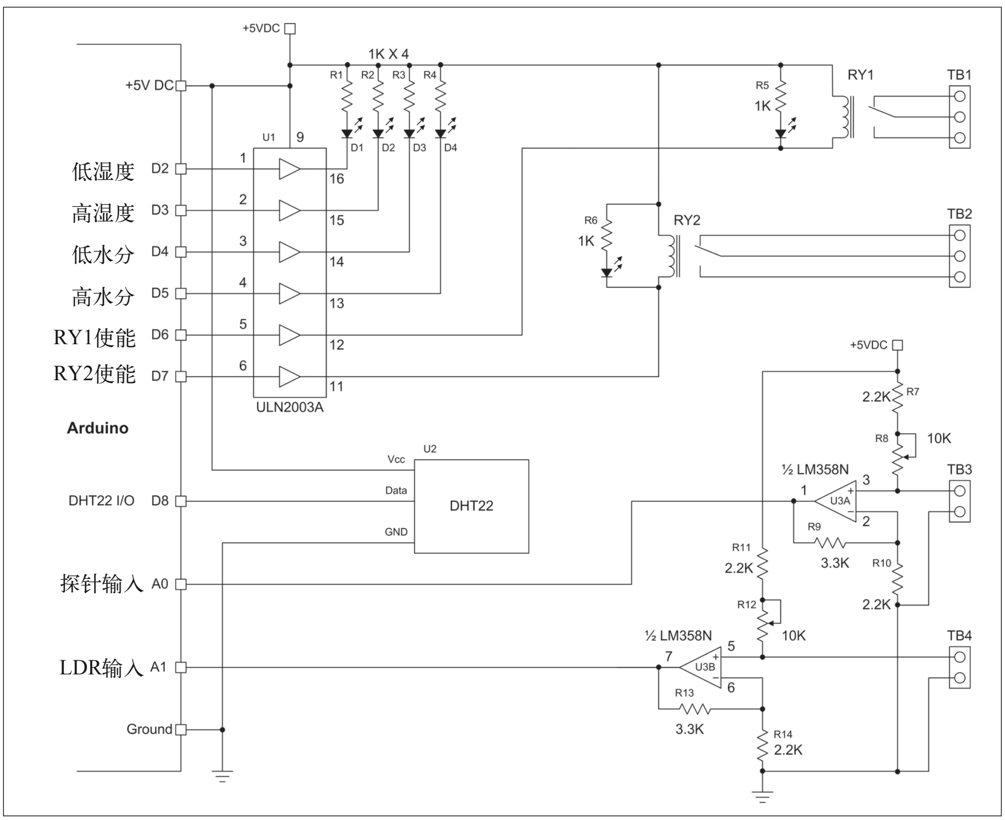


图 10-9：GreenShield 电路图

那些连接到运算放大器输入端的传感器是有效的可变电阻器，与两个微调电位器共同组成分压器。你可以通过调整微调电位器获得来自运算放大器的最佳响应，切记，不要沿着一个方向把电压调得太大。

GreenShield 采用这种设计有利于它堆叠在其他 Arduino 扩展板之上，并有效避开重要的数

字与逻辑 I/O 引脚。表 10-1 列出了 Arduino 引脚对 GreenShield 扩展板的分配情况。

表10-1：GreenShield Arduino引脚使用情况

引脚	功能	引脚	功能
D2	ULN2003A 通道 1	D7	ULN2003A 通道 6
D3	ULN2003A 通道 2	D8	DHT22 数据输入
D4	ULN2003A 通道 3	A0	土壤水分传感器输入
D5	ULN2003A 通道 4	A1	LDR 传感器输入

请注意，SPI 引脚（D10、D11、D12、D13）未使用，所以对于 SPI 扩展板，它们仍然是可用的。如果想连接 RS-232 接口并弃用 USB，D0 与 D1 也是可用的。A4 与 A5 可以作为 I2C 应用。

既然已经有了电路图，下面列出需要的全部元件，如表 10-2 所示。

表10-2：GreenShield元件列表

数量	类型	描述	数量	类型	描述
2	SRD-05VDC-SL-C	松乐 5A 继电器	4	2.2K ohm, 1/8 W	电阻器
1	DHT22	湿度 / 温度传感器	2	3.3K ohm, 1/8 W	电阻器
1	Generic	LDR 传感器	2	10K trim	PCB 载电位器
1	SainSmart	土壤水分探针	2	0.1″ (2.54 mm)	三位接线端子
6	3 mm	LED	2	0.1″ (2.54 mm)	双位接线端子
1	LM358N	运算放大器	2	0.1″ (2.54 mm)	8 位插口
1	ULN2003A	驱动 IC	2	0.1″ (2.54 mm)	6 位插口
6	1K ohm,1/8 W	电阻器	1	定制	PCB 扩展板

3. 软件

GreenShield 软件主要有 3 个功能：获取传感器输入、命令解析与输出生成、继电器功能映射。第一个功能负责从 4 个传感器输入获取数据（温度、湿度、土壤水分、环境光照度），并对数据进行存储，以提供给软件其他部分使用。命令解析功能负责解释从主 PC 发送来的命令字符串，并使用此处介绍的命令响应协议产生响应。输出功能依据传感器输入与各种命令定义的预先限制控制继电器。

主计算机与 GreenShield Arduino 之间使用的命令响应协议如表 10-3 所示。请注意，GreenShield 只对主机响应，它从来不会自己发起一个事务。

你可以使用内置于 Arduino IDE 的串行终端工具，或者退出 IDE，直接连接到 Arduino（带有 GreenShield）使用的 USB 端口。这种方法可以创建一个用户接口应用，以便设置 GreenShield 并监视其运行。实践中采用的方法是，对 Arduino 上的 GreenShield 软件进行配置，然后让它在无人看守的情形下运行。

状态查询命令

GreenShield 软件提供了 4 种查询命令，如表 10-4 所示。借助这些命令，控制主机能够获取两个继电器当前的状态（on/off）、从 LDR 或土壤传感器模拟输入最后读取的值、从 DHT222 传感器读取的最新温度与相对湿度。

表10-3：GreenShield命令响应协议（所有命令）

命令	响应	描述
AN:*n*:?	AN:*n*:*val*	获取原始 DN 中的模拟输入 *n*
GT:HMX	GT:HMX:*val*	获取湿度最大值
GT:HMN	GT:HMN:*val*	获取湿度最小值
GT:LMX	GT:LMX:*val*	获取光照最大值
GT:LMN	GT:LMN:*val*	获取光照最小值
GT:MMX	GT:MMX:*val*	获取土壤水分最大值
GT:MMN	GT:MMN:*val*	获取土壤水分最小值
GT:TMX	GT:TMX:*val*	获取温度最大值
GT:TMN	GT:TMN:*val*	获取温度最小值
HM:?	HM:*val*	返回当前湿度
RY:*n*:?	RY:*n*:*n*	返回继电器 *n* 的状态
RY:*n*:1	OK	设置继电器 *n* 为 ON
RY:*n*:0	OK	设置继电器 *n* 为 OFF
RY:A:1	OK	设置所有继电器为 ON
RY:A:0	OK	设置所有继电器为 OFF
RY:*n*:HMX	OK	如果湿度超出最大值，设置继电器 *n* 为 ON
RY:*n*:HMN	OK	如果湿度小于最小值，设置继电器 *n* 为 ON
RY:*n*:LMX	OK	如果照度超出最大值，设置继电器 *n* 为 ON
RY:*n*:LMN	OK	如果照度小于最小值，设置继电器 *n* 为 ON
RY:*n*:MMX	OK	如果土壤水分超出最大值，设置继电器 *n* 为 ON
RY:*n*:MMN	OK	如果土壤水分小于最小值，设置继电器 *n* 为 ON
RY:*n*:TMX	OK	如果温度超出最大值，设置继电器 *n* 为 ON
RY:*n*:TMN	OK	如果温度小于最小值，设置继电器 *n* 为 ON
ST:HMX:*val*	OK	设置湿度最大值
ST:HMN:*val*	OK	设置湿度最小值
ST:LMX:*val*	OK	设置照度最大值
ST:LMN:*val*	OK	设置照度最小值
ST:MMX:*val*	OK	设置土壤水分最大值
ST:MMN:*val*	OK	设置土壤水分最小值
ST:TMX:*val*	OK	设置温度最大值
ST:TMN:*val*	OK	设置温度最小值
TM:?	TM:*val*	返回当前温度

表10-4：GreenShield查询命令

命令	响应	描述
RY:*n*:?	RS:*n*:*n*	返回继电器 *n* 的状态
AN:*n*:?	AN:*n*:*val*	获取原始 DN 中的模拟输入 *n*
TM:?	TM:*val*	返回最新 DHT22 温度
HM:?	HM:*val*	返回最新 DHT22 湿度

继电器超控命令

你可以使用软件命令控制 GreenShield 上的继电器。表 10-5 列出了继电器超控命令，共有 4 种。通过这些命令可以把单个继电器设置为 on 或 off，也可以一次性把两个继电器设置为 on 或 off。

> 请注意，使用超控命令设置继电器时，之前的任何设定值映射都会被删除。为了把设定值再次应用到继电器，需要将一个设定值命令发送到 GreenShield。

表10-5：GreenShield继电器命令

命令	响应	描述
`RY:n:1`	`OK`	设置继电器 *n* 为 ON
`RY:n:0`	`OK`	设置继电器 *n* 为 OFF
`RY:A:1`	`OK`	设置所有继电器为 ON
`RY:A:0`	`OK`	设置所有继电器为 OFF

继电器动作映射命令

湿度、照度、土壤含水量、环境温度的最小或最大设定值条件都可以与某个继电器的激活行为映射。表 10-6 列出了继电器的设定值命令。将一个动作映射到继电器时，最新的映射命令将覆盖掉之前的任何命令。

表10-6：GreenShield继电器设定值命令

命令	响应	描述
`RY:n:HMX`	`OK`	当湿度大于或等于 max 时，设置继电器 *n* 为 ON
`RY:n:HMN`	`OK`	当湿度小于或等于 min 时，设置继电器 *n* 为 ON
`RY:n:LMX`	`OK`	当照度大于或等于 max 时，设置继电器 *n* 为 ON
`RY:n:LMN`	`OK`	当照度小于或等于 min 时，设置继电器 *n* 为 ON
`RY:n:MMX`	`OK`	当土壤含水量大于或等于 max 时，设置继电器 *n* 为 ON
`RY:n:MMN`	`OK`	当土壤含水量小于或等于 min 时，设置继电器 *n* 为 ON
`RY:n:TMX`	`OK`	当温度大于或等于 max 时，设置继电器 *n* 为 ON
`RY:n:TMN`	`OK`	当温度小于或等于 min 时，设置继电器 *n* 为 ON

设定值命令

使用表 10-7 列出的 ST 命令可以定义最小与最大设定值，使用 GT 命令可以把设定值返回给主控 PC。你可以在任何时候修改设定值。

继电器功能映射将继电器与传感器输入以及一组状态改变条件（以上下限的形式存在）关联起来。如果传感器值高于或低于主控系统设置的限制值，继电器就会启动工作。继电器关联不是排他性的，这意味着两个继电器可以同时指派为同一个传感器的输入与限制条件。这样做可能没有什么意义，但其实是可以的。

表10-7：GreenShield min/max设置命令

命令	响应	描述
ST:HMX:*val*	OK	设置湿度最大值
ST:HMN:*val*	OK	设置湿度最小值
ST:LMX:*val*	OK	设置照度最大值
ST:LMN:*val*	OK	设置照度最小值
ST:MMX:*val*	OK	设置土壤含水量最大值
ST:MMN:*val*	OK	设置土壤含水量最小值
ST:TMX:*val*	OK	设置温度最大值
ST:TMN:*val*	OK	设置温度最小值
GT:HMX	GT:HMX:*val*	获取湿度最大值
GT:HMN	GT:HMN:*val*	获取湿度最小值
GT:LMX	GT:LMX:*val*	获取照度最大值
GT:LMN	GT:LMN:*val*	获取照度最小值
GT:MMX	GT:MMX:*val*	获取土壤含水量最大值
GT:MMN	GT:MMN:*val*	获取土壤含水量最小值
GT:TMX	GT:TMX:*val*	获取温度最大值
GT:TMN	GT:TMN:*val*	获取温度最小值

尽管 GreenShield 目前配备了两个继电器，但其实对可使用的继电器数量没有硬性限制。从表 10-1 中可以看到，I2C 引脚（A4 与 A5）是可用的，因此可以使用 I2C 数字 I/O 扩展板把更多设备连接到 Arduino 开发板。

10.3.4 制作原型

下面我将使用 Duinokit 工具为本项目制作原型，如图 10-10 所示。Duinokit 是一套“聪明”的工具，包括一组传感器、LED 灯、开关，以及其他 Arduino Nano 配件。它们全部安装于一个大的 PCB 上，对外提供大量插口。Duinokit 工具也提供了安装传统扩展板（或者扩展板堆叠）的位置。老式多功能的电子项目套装曾风靡一时，Duinokit 像是一个现代版的多功能电子项目套装。



图 10-10：Duinokit 工具

另一个同样有效的方法是，从传感器套装与几乎任何 Arduino 组装所有需要的元件。但相比之下，使用 Duinokit 工具会让整个过程变得更加整洁、干净。此外，Duinokit 工具也提供了友好的开发平台，等待 PCB 与其他部件到来期间，你可以使用它编写软件。从 http://duinokit.com 与 Amazon.com 网站上，你可以购买到 Duinokit 工具。

Duinokit 带有一个 DHT11 温湿度传感器，它比 DHT22 速度更慢、分辨率更低（GreenShield 将采用 DHT22 传感器）。在软件方面，DHT11 与 DHT22 类似，但不完全一样。不同于 DHT11，DHT22 传感器使用不同的数据字（位串），以满足更高精度的要求。

LDR 与土壤水分传感器采用一个 LM358 双运算放大器，每个输入分配一半。我把 LM358 安装在 Duinokit 提供的免焊面包板上。Duinokit 上也提供了安装电阻、两个 10 K 电位器的位置，用作输入偏移调整控制。

1. 原型软件

我们将在原型与最终产品中开发软件。第一步是编写运行在 Duinokit 原型上的软件，用于读取传感器输入，检验 LM358 运算放大器电路是否如预期那样工作，并支持一些测试判断初始输入范围限制。下一步是开发最终软件，它支持命令响应协议（参考 10.3.3 节中的“软件”部分的定义），以及继电器功能映射。实际的扩展板硬件将被用于开发最终软件。

原型软件从模拟输入与板载 DHT11 传感器读取数据。初始输入范围确定时，将使用这些数据对原型进行测试。输出显示在 Arduino IDE 提供的串口监视器窗口。示例 10-1 中的原型测试软件包含于一个名为 gs_proto.ino 的程序文件。

> 示例 10-1 函数中涉及的各种温度转换与露点值对于基本的 GreenShield 不是必需的。我把它们作为例子包含进来，如果需要，你可以使用。

示例 10-1　GreenShield 传感器原型软件

```
// GreenShield原型软件
// 2015.J.M.Hughes
//
// 使用DHT11库,该作者是George Hadjikyriacou、SimKard、Rob Tillaart
//
// 不断重复读取与输出温度、湿度、土壤水分、光照水平。不带继电器设置点功能。

#include <dht11.h>

// 定义

#define LHLED       2   // D2 低湿度LED
#define HHLED       3   // D3 高湿度LED
#define LMLED       4   // D4 低水分
#define HMLED       5   // D5 高水分
#define RY1OUT      6   // D6 RY1使能
#define RY2OUT      7   // D7 RY2使能
#define DHT11PIN    8   // D8 DHT11数据
#define SMSINPUT    A0  // A0 SMS输入
```

```
#define LDRINPUT   A1  // A1 LDR输入

// 全局变量

int curr_temp = 0;    // 当前(最近)温度
int curr_hum  = 0;    // 当前湿度
int curr_ambl = 0;    // 当前环境光强度
int curr_sms  = 0;    // 当前土壤水分

int cntr = 0;


// dht11是全局对象
dht11 DHT11;


// 从DHT11读取数据并存储到curr_hum与curr_temp全局变量
// 以后要使用的变量
void readDHT()
{
    if (!DHT11.read(DHT11PIN)) {
        curr_hum = DHT11.humidity;
        curr_temp = DHT11.temperature;
    }
}


// 通过LM358 op放大器电路从模拟输入读取数据,并存储到curr_ambl与curr_sms,供以后使用
void readAnalog()
{
    curr_ambl = analogRead(LDRINPUT);
    curr_sms = analogRead(SMSINPUT);
}


// 从℃转换为℉
// 示例代码在http://playground.arduino.cc/main/DHT11Lib
double Fahrenheit(double celsius)
{
    return 1.8 * celsius + 32;
}


// 从℃转换为T(开氏温度)
// 示例代码在http://playground.arduino.cc/main/DHT11Lib
double Kelvin(double celsius)
{
    return celsius + 273.15;
}


// dewPoint()函数 NOAA
// 参考:http://wahiduddin.net/calc/density_algorithms.htm
// 示例代码在http://playground.arduino.cc/main/DHT11Lib
```

```
double dewPoint(double celsius, double humidity)
{
    double A0= 373.15/(273.15 + celsius);
    double SUM = -7.90298 * (A0-1);
    SUM += 5.02808 * log10(A0);
    SUM += -1.3816e-7 * (pow(10, (11.344*(1-1/A0)))-1) ;
    SUM += 8.1328e-3 * (pow(10,(-3.49149*(A0-1)))-1) ;
    SUM += log10(1013.246);
    double VP = pow(10, SUM-3) * humidity;
    double T = log(VP/0.61078); // temp var
    return (241.88 * T) / (17.558-T);
}


// delta max=0.6544 wrt dewPoint()
// 比dewPoint()快5x
// 参考:http://en.wikipedia.org/wiki/Dew_point
// 示例代码在http://playground.arduino.cc/main/DHT11Lib
double dewPointFast(double celsius, double humidity)
{
    double a = 17.271;
    double b = 237.7;
    double temp = (a * celsius) / (b + celsius) + log(humidity/100);
    double Td = (b * temp) / (a - temp);
    return Td;
}


void setup()
{
    // 初始化串口I/O
    Serial.begin(9600);

    // 初始化当前数据变量
    pinMode(LHLED,OUTPUT);
    pinMode(HHLED,OUTPUT);
    pinMode(LMLED,OUTPUT);
    pinMode(HMLED,OUTPUT);
    pinMode(RY1OUT,OUTPUT);
    pinMode(RY2OUT,OUTPUT);

    // 设置AVR引脚
    curr_temp = 0;
    curr_hum  = 0;
    curr_ambl = 0;
    curr_sms  = 0;
}


void loop()
{
    // 获取DHT11读数
    readDHT();

    // 获取LDR与SMS读数
```

```
    readAnalog();

    // 打印数据到输出
    Serial.println("\n");
    Serial.print("Raw LDR           : ");
    Serial.println(curr_ambl);
    Serial.print("Raw SMS           : ");
    Serial.println(curr_sms);
    Serial.print("Humidity (%)      : ");
    Serial.println((float)DHT11.humidity, 2);
    Serial.print("Temperature (oC)  : ");
    Serial.println((float)DHT11.temperature, 2);
    Serial.print("Temperature (oF)  : ");
    Serial.println(Fahrenheit(DHT11.temperature), 2);
    Serial.print("Temperature (K)   : ");
    Serial.println(Kelvin(DHT11.temperature), 2);
    Serial.print("Dew Point (oC)    : ");
    Serial.println(dewPoint(DHT11.temperature, DHT11.humidity));
    Serial.print("Dew PointFast (oC): ");
    Serial.println(dewPointFast(DHT11.temperature, DHT11.humidity));

    // 向上滚动对齐显示
    for (int i = 0; i < 12; i++)
        Serial.println();

    delay(1000);
}
```

上述代码中，`setup()` 函数用于打开并初始化串口 I/O，设置引脚模式，以及为当前数据读取清空全局变量。每次调用 `readDHT()` 与 `readAnalog()` 函数都会从 DHT11 与模拟输入获取最新数据值，并把它们存入相应变量。

主循环用于读取模拟输入与 DHT11 数据，对数据进行格式化，并把当前值写到 USB 串口监视器。它不会与主控计算机进行通信，并且不做任何继电器设定值映射，其目的在于连续获取并显示传感器数据。

原型为 DHT11 采用一个开源库。最终版本将为 DHT22 使用定制库，但对于原型而言，它不是必需的。DHT11 库由 George Hadjikyriacou、SimKard、Rob Tillaart 编写，你可以从 Arduino Playground（http://bit.ly/apg-dht11）下载使用。`Fahrenheit()`、`Kelvin()`、`dewPoint()`、`dewPointFast()` 函数来自同一源代码库。

2. 原型测试

使用 Duinokit，我们能测试 GreenShield 的各种传感器输入功能，并对运行做优化调整。如果电路有问题（由于电路很简单，所以发现的问题也很容易解决），你可以在原型测试中发现并修正。一个 PCB 制作好并安装元件后，再尝试修正其上的问题会让人感到很懊恼，并且这一过程中总是存在损坏正常元件的风险。

对于 GreenShield 来说，我们想检查其传感器是否正常工作，软件能否从土壤水分传感器、LDR 等获取有意义的数据值，以及温湿度传感器能否像预期的那样工作。为此，我们将使用一些干燥的沙子、水、可靠的数字温度计、冰箱、烤箱，还需要晴好的天气条件。

首先要检测的是温湿度传感器。准备一个外部温度计（我使用数字温度计，在长引线上带有传感器），开始读入环境数据。接着将 Duinokit 与温度计放入冰箱。一个小的电脑提供电源并显示温度，USB 电缆很细，不影响冰箱门正常关闭。最后，把 Duinokit 与温度计放入大约 140 ℉（60 ℃）的烤箱。借助 3 个数据点，我们能够生成一条粗糙的校准曲线，对温度传感器的偏差进行补偿。

测试湿度响应有点棘手，但在可用范围端点附近获取读数并不太难。在冰箱冷冻室的短暂停留将把传感器暴露在低湿度环境中。冷冻室很干燥，因为空气中的水分都凝结在冷凝管上。因此，若食物在冷冻之前密封不当，冷冻后就会形成冻斑。这也是冷冻干燥背后的原理，常见于更低温度（大约 −112 ℉或 −80 ℃）与部分真空环境中。就厨房的冷冻柜来说，我们希望看到的湿度大约是 5% 或更低一些。

> 另一种方法就是所谓的“盐雾试验”。该技术使用水饱和盐在封闭环境中建立恒定相对湿度。在 Ambient Weather wiki（http://bit.ly/wiki-aw）上，你可以使用一种方法进行盐基校准。如果打算这样做，请注意不要把任何盐分或水分洒到电路元件上。对于像 Duinokit 这样的“大个头”，这种方法可能不太实用，但该方法可以用在制作好的 GreenShield 上。

有了一个低湿度读数之后，接下来要在炉子上烧水，并使用小风扇把蒸汽吹到传感器之上。这样产生的空气流不会完全湿透，湿度范围为 80%~90%。这些测试可以用来检测传感器是否正常工作，但其实我们并不能把数据应用到其他东西上，因为没有用以进行参考、比较的基准。如果你有一个精确的湿度传感器，那么一定要用它建立一条校准曲线，类似于我们前面为温度创建的校准曲线。

测试土壤水分传感器时，要用到一些干燥的沙子、一个天平、一些水。首先，找一个大的玻璃瓶或陶瓷碗。从中选择任意一种容器，但保证要能够盛下大约 1 L 水。做这项测试时，请不要使用金属碗，因为水分探针有电流通过，使用金属碗可能产生错误读数。首先，为容器称重并记录数值，稍后会用到。接着，量出大约 225 g 沙子，倒入容器，然后把容器再次放到天平上称重。沙子的实际重量等于天平显示的重量减去容器的重量。如果愿意，你可以把容器留在天平上，以便完成其余测试步骤。

现在把土壤水分探针插入干燥的沙子，留意 Arduino IDE 串口监视器窗口显示的读数。然后移走传感器，不断向沙子注水，直到总重量比沙子与容器重量之和 1 L 左右。稍等片刻，让水分浸入沙子。此时沙子摸上去应该湿湿的，但又不致于太过潮湿。

再次插入传感器，观察输出。现在沙子的饱和度约为 50%，从干、湿两组读数可知，我们能够在两者之间插入一个表示 25% 湿度的点。

最后，测试 LDR 光电管。其实，光电管的响应不那么重要，但最好还是建立一个低光跳变点（trip point）。阴天时，GreenShield 通过光电管感知周围光线微弱，进而打开辅助照明，或者通过光电管判断是白天还是黑夜。测试光电管只需要你房子里的一个内部空间（窗帘半开，且没有灯光），以及室外晴好的天气。室外直射的阳光使光电管暴露在大量光线照射之下，房子的内部空间营造了阴暗环境，其光照程度大致相当于室外的多云天气。

我们需要记录来自温湿度传感器、LDR、土壤水分探针的数据。第一次安装 GreenShield

时，这些数据将被用作初始值。由于现在已经知道了相关数据，所以开始时就无需再猜测应当将最小与最大设定值设为多少。

10.3.5 最终软件

原型软件只处理传感器输入。除此之外，最终软件还要控制主控接口，对继电器设定值功能进行映射。该过程将涉及输入命令解析、数据存储与查询等处理。

除了 4 个湿度与温度范围状态 LED 灯之外，没有其他输出显示，也没有手动控制输入。简单的 USB 串口在 GreenShield Arduino 与主控计算机之间做命令响应事务。大部分软件解释来自主控 PC 的命令，并将设定值映射到继电器。

1. 组织源代码

GreenShield 最终软件的源代码包含在多个源文件或模块中。在 Arduino IDE 中打开主文件 GreenShield.ino 时，它也会把位于同一目录下的其他关联文件一同打开。这些次要文件在 IDE 中以标签（tab）的形式展现，如图 10-11 所示。

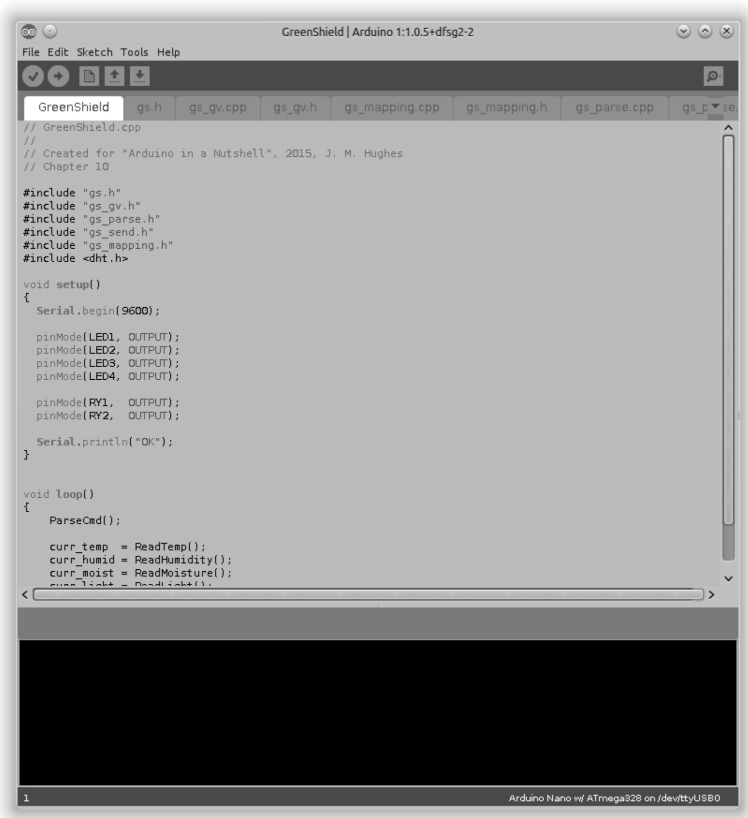


图 10-11：在 Arduino IDE 中打开 GreenShield 文件

表 10-8 列出了 GreenShield 的所有文件。其中，gs.h 与 gs_gv.h 两个文件被所有源代码模块共享。在 gs_gv.cpp 中定义的全局变量（在常规程序的开始部分会经常看到它们）被单独进行编译，并于必要时在其他模块间进行分享。

表10-8：GreenShield源代码模块

模块	功能
GreenShield.ino	主模块，包含 `setup()` 与 `loop()`
gs_gv.cpp	全局变量
gs_gv.h	包含文件
gs.h	常量定义（`#define` 语句）
gs_mapping.cpp	功能映射
gs_mapping.h	包含文件
gs_parse.cpp	命令解析
gs_parse.h	包含文件
gs_send.cpp	向主机发送数据功能
gs_send.h	包含文件

采用这种方式组织项目后，修改代码的某一个部分时，不必再逐行浏览源代码，这使得修改代码变得更简单。一旦主要部分修改完成，那么这些改变可以轻松应用到其他模块，而不会对最终代码产生干扰。这种方法也有利于我们站在模块角度思考软件，使理解与维护软件变得更容易。

2. 软件描述

图 10-12 显示了软件主循环的流程图。`loop()` 函数从 Start 方框开始执行，一直到 Arduino 断电为止。请注意，图中划分了 3 个主要功能区域，分别是命令输入与响应处理、数据采集、最小 / 最大设定值测试。此外，还要注意带有 Setpoint Test 的区块，是 4 个区块的一个实例，每对 min/max 设定值对应一个。为了让流程图大小合理，图中只显示一个测试部分。

示例 10-2 中的 GreenShield.ino 源文件包含 `setup()` 与 `loop()` 两个函数。可以在 GitHub 上（https://github.com/ardnut）找到完整的 GreenShield 软件。

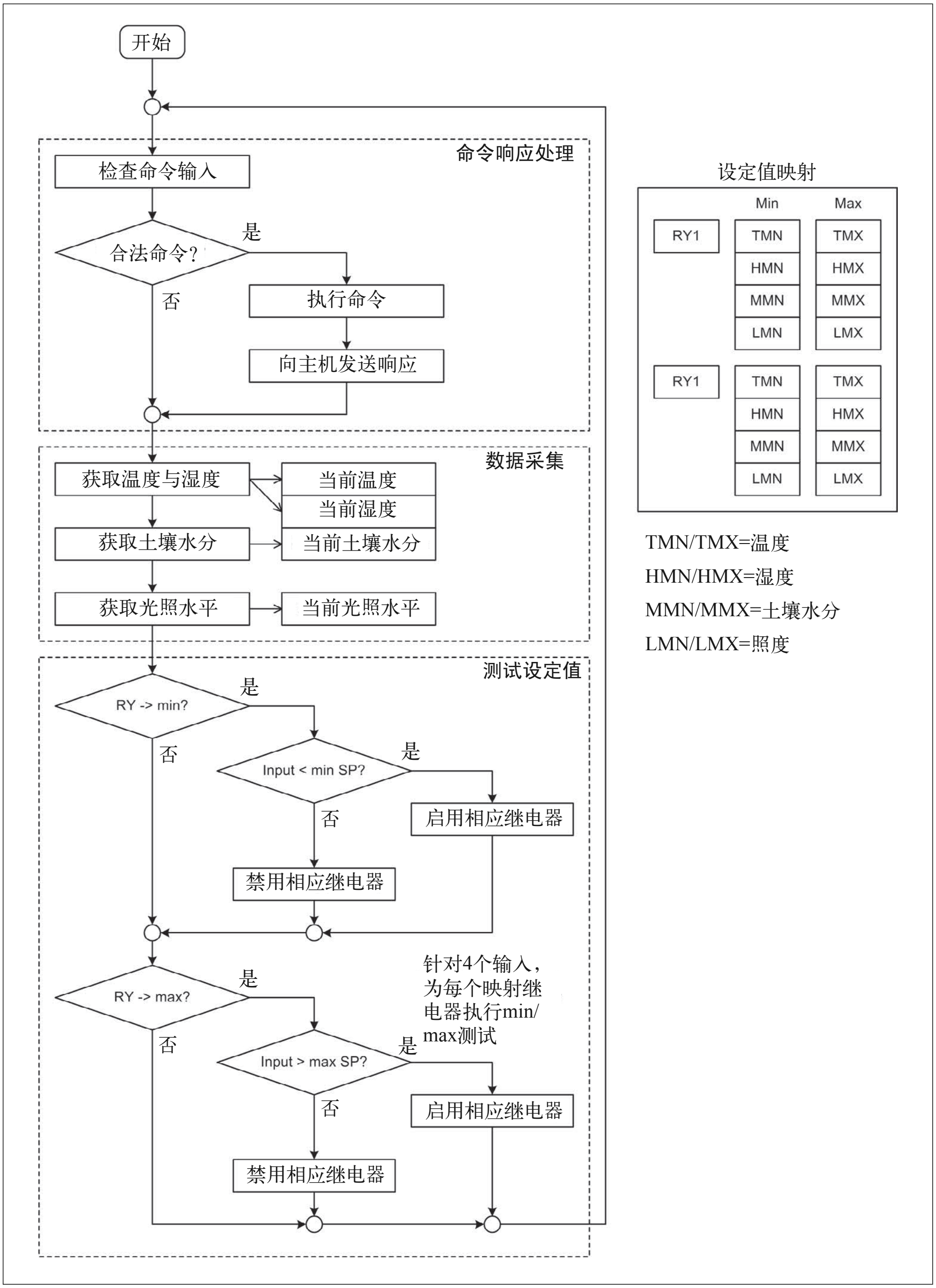


图 10-12：GreenShield 流程图

示例 10-2　GreenShield 主文件

```
// GreenShield.ino
//
// 创建于2016,J.M.Hughes
// 第10章

#include "gs.h"
#include "gs_gv.h"
#include "gs_parse.h"
#include "gs_send.h"
#include "gs_mapping.h"
#include <dht.h>

void setup()
{
  Serial.begin(9600);

  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  pinMode(LED3, OUTPUT);
  pinMode(LED4, OUTPUT);

  pinMode(RY1, OUTPUT);
  pinMode(RY2, OUTPUT);

  Serial.println("OK");
}


void loop()
{
    ParseCmd();

    curr_temp = ReadTemp();
    curr_humid = ReadHumidity();
    curr_moist = ReadMoisture();
    curr_light = ReadLight();

    ScanMap();
}
```

Arduino IDE 根据 `#include` 语句判断哪些模块属于该代码集。即使顶层模块没有直接使用某个源文件，你仍然要把它包含进来。

示例 10-3 显示的是全局定义文件 gs.h，其中定义了一系列常量，GreenShield 源代码模块会用到它们。`#define` 语句会产生更小的编译对象，5.4.7 节已经做出说明。

示例 10-3　GreenShield 全局定义

```
// gs.h
//
// 创建于2016,J.M.Hughes
// 第10章

#ifndef GSDEFS_H
#define GSDEFS_H
```

```
#define MAXINSZ       12       // 输入缓冲大小

#define NOERR         0        // 错误代码
#define TIMEOUT       1
#define BADCHAR       2
#define BADVAL        3

#define LED1          2        // LED引脚定义
#define LED2          3
#define LED3          4
#define LED4          5
#define RY1           6        // 继电器引脚定义
#define RY2           7

#define DHT22         8        // DHT22 I/O引脚
#define MPROBE        A0       // 土壤探针输入
#define LDRIN         A1       // LDR输入

#define MAXRY         2        // 继电器最大数

#define MAP_NONE      0        // 映射向量
#define MAP_TEMPMIN   1
#define MAP_TEMPMAX   2
#define MAP_HUMIDMIN  3
#define MAP_HUMIDMAX  4
#define MAP_MOISTMIN  5
#define MAP_MOISTMAX  6
#define MAP_LIGHTMIN  7
#define MAP_LIGHTMAX  8

#endif
```

示例 10-3 中，继电器的数量 `MAXRY` 定义为 `2`。如果硬件可以支持更多继电器，你可以将其修改为一个更大的值，并且输出也不必是继电器。示例 10-4 显示的是 gs_mapping.h 包含文件，其中声明了一系列函数，用于读取 DHT22 与模拟输入，设置每个继电器（或者所有继电器）的 on/off 状态，控制状态指示灯 LED，扫描响应条件并控制继电器（使用 `ScanMap()` 函数）。

示例 10-4　GreenShield 映射函数

```
//gs_mapping.h
//
// 创建于2016,J.M.Hughes
// 第10章

#ifndef GSMAP_H
#define GSMAP_H

void ReadDHT22();
int  ReadTemp();
int  ReadHumidity();
int  ReadMoisture();
int  ReadLight();

int  RyGet(int ry);
```

```
void RySet(int ry, int state);
void RyAll(int state);

void LEDControl(int LEDidx);
void ScanMap();

#endif
```

示例 10-5 显示的是 `ScanMap()` 函数，它位于 gs_mapping.cpp 文件，每当执行 GreenShield.ino 主文件中的 `loop()` 函数时，都会调用执行，它会根据相应条件开启或关闭继电器。

示例 10-5　GreenShield 函数映射扫描器

```
// 注意:本代码并不对是否有多个继电器映射到相同工作模式进行检测,也未禁止这种情况发生
// 每个RY都被映射到8种可能的工作模式之一。
// 确定对某个特定继电器的映射,并查看是否有使能条件出现。
// 这可以推广到任何合理数量的继电器。
void ScanMap()
{
    for (int i = 0; i < MAXRY; i++) {
        if (rymap[i] != MAP_NONE) {
            switch (rymap[i]) {
                case MAP_TEMPMIN:
                   if (curr_temp < mintemp)
                       { RySet(i, 1); } else { RySet(i, 0); }
                   break;
                case MAP_TEMPMAX:
                   if (curr_temp > maxtemp)
                       { RySet(i, 1); } else { RySet(i, 0); }
                   break;
                case MAP_HUMIDMIN:
                   if (curr_humid < minhum)
                       { RySet(i, 1); } else { RySet(i, 0); }
                   break;
                case MAP_HUMIDMAX:
                   if (curr_humid > maxhum)
                       { RySet(i, 1); } else { RySet(i, 0); }
                   break;
                case MAP_MOISTMIN:
                   if (curr_moist < minmoist)
                       { RySet(i, 1); } else { RySet(i, 0); }
                   break;
                case MAP_MOISTMAX:
                   if (curr_moist > maxmoist)
                       { RySet(i, 1); } else { RySet(i, 0); }
                   break;
                case MAP_LIGHTMIN:
                   if (curr_light < minlite)
                       { RySet(i, 1); } else { RySet(i, 0); }
                   break;
                case MAP_LIGHTMAX:
                   if (curr_light > maxlite)
                       { RySet(i, 1); } else { RySet(i, 0); }
                   break;
```

```
                default:
                    // Do nothing
                    break;
            }
        }
    }
}
```

源代码模块 gs_parse.cpp 中包含一个主要函数 ParseCmd()，以及几个辅助函数（CntInt()、SendErr()）。示例 10-6 显示了 gs_parse.h 文件的内容。

示例 10-6　GreenShield 解析模块的包含文件

```
// gs_parse.h
//
// 创建于2016,J.M.Hughes
// 第10章

#ifndef GSPARSE_H
#define GSPARSE_H

void ParseCmd();
int CvtInt(char *strval, int strt, int strlen);
void SendErr(int errtype);

#endif
```

ParseCmd() 函数是 GreenShield 软件中代码最长的函数之一。如果采用快速条件递减的树状解析器判断输入命令的类型，那就需要提取子函数代码与所有参数。该函数也会执行即时命令，比如开启或关闭继电器、返回继电器状态信息、获取并返回模拟数据到主控机器或用户。到达递减树结构的终结点时，就会执行即时命令。

10.3.6　装配

有关绘制电路图与印制电路板的详细内容，已经超出本书讨论的范围。此处只对从电路图到 PCB 布局设计，再到制作好 PCB 的过程中涉及的步骤进行大致讲解。更多底层细节，我建议你参考附录 D。你也可以使用谷歌搜索查找 CadSoft Eagle，获取大量使用教程。不过，我建议你阅读 SparkFun、Adafruit、CadSoft 网站（http://www.cadsoftusa.com）中的相关教程。

图 10-13 显示的是 GreenShield 的电路图（Eagle 版本）。请注意标注有 ARDUINO_R3_SHIELD 的部分，它来自 SparkFun 元件库，专门用于制作扩展板。与后面在 PCB 布局设计阶段采用手工方式设置排针垫相比，使用它要方便得多。

像使用其他工具一样，你需要花一些时间适应 Eagle 中的电路图编辑器。有时它并不那么直观易用。通常，我会使用其他工具创建具有出版级质量的线稿，包括电路图。而创建 PCB 时，最好使用电路图编辑器与 PCB 布局设计工具，方便共享数据。你可以把图 10-13 与图 10-9 进行比较，观察它们之间有何不同。Eagle 工具与 Switchinator 项目中使用的 Fritzing 设计工具都能让电路图与布局设计保持同步，而用于绘制线稿与插图的图形工具则无法做到这一点。

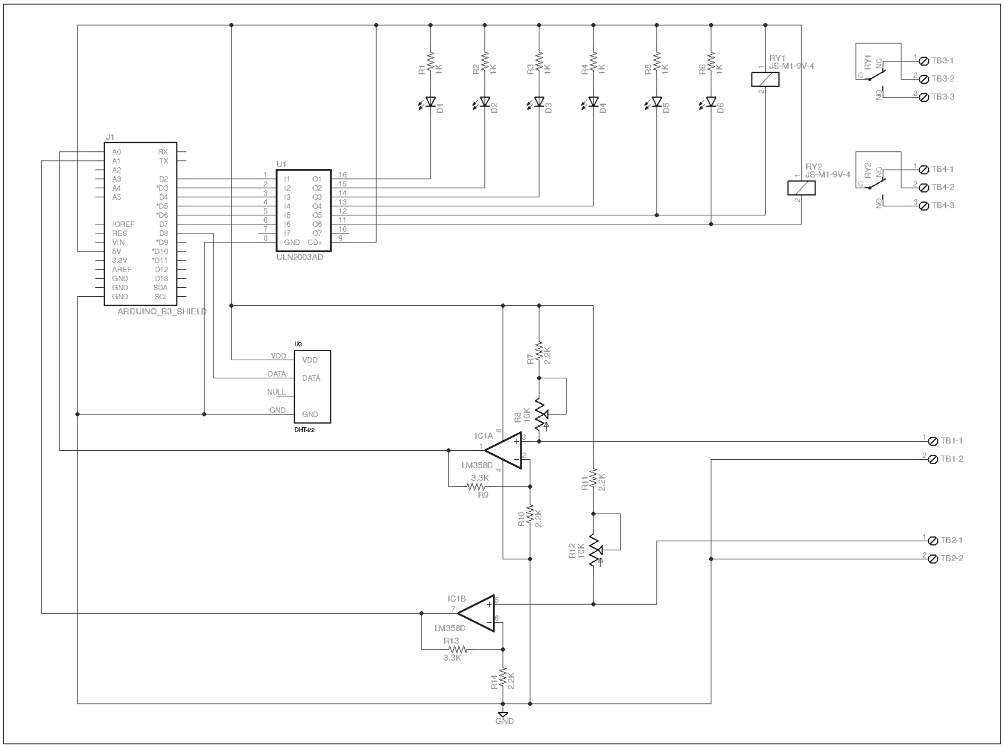


图 10-13：GreenShield 电路图（Eagle 版本）

创建电路图时，请确保选择了正确的元件。比如，对于电阻，不论是 0805SMD 元件还是 1/4 W 穿孔元件，它们的符号都是一样的。有时，从电路图编辑器的元件库中找到正确的元件并非易事。在 Egale 中，你可以使用通配符搜索元件库。比如为 GreenShield 查找 ULN2003AD 元件时，先在搜索框中输入 *2003*，然后在 uln-udn 类别下即可找到。有时需要上网搜索要使用的元件，这些元件可能是别人可能已经制作好的。SparkFun、Adafruit 以及其他厂商创建了大量元件库，你可以免费下载并使用。

制作好电路图后，就可以生成 PCB 了。最初，PCB 布局只是一大堆元件与细的连接线（有时也称为“架空线”），形成“鼠巢”式布线。它是点对点的连接，在从电路图创建的网络表文件中定义。

首先将所有元件移入 PCB 区域，然后认真安排它们。安排这些元件时，主要目标是把连接器定位到需要的位置上，并根据功能对元件进行分组，必要时通过旋转元件尽量减少出现跨越“鼠巢”的连接线。这个初始步骤完成后，再开始创建连接各个元件的迹线就会容易得多。

你可以选择手工布线，以鼠巢线作为指引（它们有时也称为“轨道线”）。如果自我感觉运气不错，可以尝试使用自动布线器帮助布线。Eagle 也有一个自动布线器，但我一般不用。通常，自动布线是一个迭代过程，不断尝试得到好的布局，丢弃不好的布局，移动并旋转各个元件，而后再次尝试。不久你会发现，对于某些设计，采用手工方式布线效率更高。

在 Eagle 中，你不必手工设置过孔（过孔通过镀通孔把迹线从 PCB 的一面转移到另一面，因此得名）。如果需要从 PCB 顶部转移到底部（反之亦然），迹线绘制工具条左侧有层选择下拉菜单，使用它可以很简单地从一面换到另一面。Eagle 会在合适位置自动设置过孔，并把迹线布线切换到 PCB 上所选的一面。

图 10-14 是 PCB 布线图。顶面（带有元件的一面）是红棕色，底面（焊接面）是蓝色。如果你正看的是本书纸质版，将会看到顶面迹线呈现为亮灰色，底面迹线呈现深灰色。元件轮廓为淡灰色。

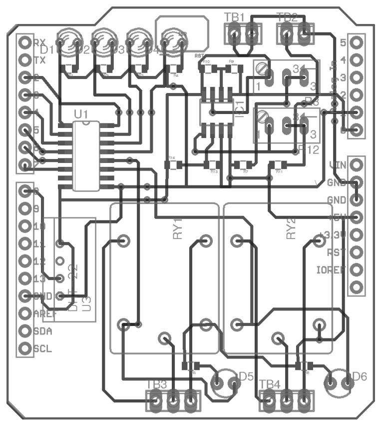


图 10-14：GreenShield PCB 布线

由 Eagle CAM（计算机辅助制造）工具生成的文件称为 Gerber 文件，PCB 制造商使用它制造实际的 PCB。我使用 Gerbv 工具（gEDA 包的一部分）加载并查看 Gerber 文件，进行最后检查。图 10-15 是 Gerbv 工具运行时的屏幕截图。

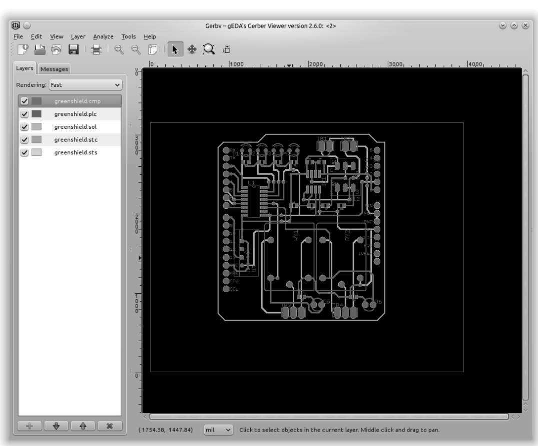


图 10-15：Gerbv Gerber 文件查看器

为了充分利用原型 PCB 的低廉服务，GreenShield 板做成了矩形。这看上去可能显得有点古怪，但它不会影响板子正常工作。在图片中你不会看到这种外形，但最初所做的布局采用的是 Arduino 扩展板的外形。把转角转换为直角花了大约 5 min，并且只在即将送去制造之前才这样做。如果这真的很重要，我会多花一些钱请人做边缘布线与修整处理，或者我会使用自己的布线器亲自动手处理。我选择让它保持矩形。

从制造商那里拿到制作好的 PCB 大约要 7~10 天，图 10-16 是制造商发来的裸 PCB。

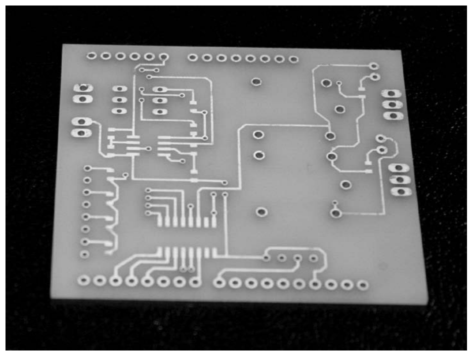


图 10-16：制作好的 GreenShield PCB 裸板

把各种元件焊接到 PCB 上后，最好花几分钟做一些检查，仔细查看是否有冷焊点，以及焊盘与迹线之间是否有短路（锡桥）发生。检查时，我会使用标准的珠宝放大镜。图 10-17 显示的是最后组装好的 GreenShield 扩展板。

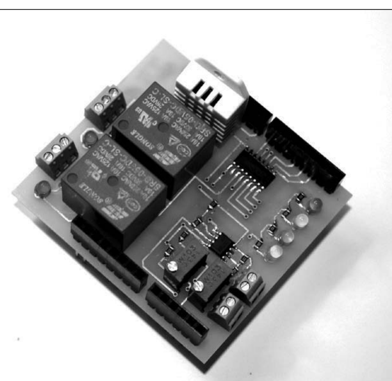


图 10-17：组装好的 GreenShield PCB

你可能已经注意到，我使用排针连接底部的 Arduino 或其他扩展板。虽然把另一个扩展板放置到 GreenShield 之上可能会有些不便（我不建议这样做），但我想对 I/O 做一些简单的可用性测试。

10.3.7 最终验收测试

表面贴装元件有一系列特有的潜在问题。为 GreenShield 上电之前，我们应该做些检查，保证所有元件正常连接，且 PCB 上也不存在短路问题。

目视检查

仔细观察 PCB 上的元件，检查是否存在锡桥（多发生在焊盘之间，或者焊盘与迹线之间）。认真观察电阻，查看引脚是否有空焊造成未连接的情况。使用焊料与焊枪进行焊接时，常常会发生这样的问题（我使用焊膏与热风 SMD 回流焊工具）。检查 IC 上的焊盘（U1 与 IC1），确保它们之间不存在锡桥，防止出现短路问题。

元件布局

有 9 个元件可能会被意外装反，它们是两个 IC、LED 灯、DHT22 传感器模块。运算放大器 IC 上有一个凹痕或圆点，用于指示引脚 1，但有些封装会采用斜坡边进行指示。除了引脚长度不同外，在靠近阴极连接处，LED 通常会有一块小的平整区。

短路

使用 DMM（最好带有连续测试功能，蜂鸣器模式），检查 LM358 运算放大器（IC1）的每对引脚是否存在短路问题，确保没有引脚与其他引脚发生短路。接着检查 IC1 引脚 8 上的 VCC 是否已经连到排针的 5 V 电源上。此外，还要检查引脚 4 是否连接到排针的地线引脚上。

重复上述过程，检查 ULN2003A 驱动器（U1），确保输入、输出引脚未发生短路，引脚 8 连接到地线，引脚 9 连接到 5 V 电源。

电力安全

将 GreenSield 连接到 Arduino 开发板之前，使用 DMM 测量排针上接地引脚与 +5 V 引脚之间的电阻。你应该能够看到一个不小于 30 K Ω 的阻值，也有可能更大一些。如果看到的读数为 0，则表示有短路发生。为 GreenShield 上电之前，必须找到并且解决这些短路问题。

如果 GreenShield 顺利通过上面一系列电气测试，接下来要把它安装到 Arduino 上，并为其供电。初始状态下，GreenShield 扩展板上应该没有 LED 灯点亮（除非某个控制着数字 I/O 引脚的软件已经运行在 Arduino 的 AVR 上）。功能测试有如下 4 个基本步骤。

最初功能测试

功能测试中首先要做的是，把原型测试中进行的测试重新做一遍，对模拟输入与 DHT22 进行测试。

把原型版本软件上传到 Arduino，并打开 IDE 的串口监视器窗口。若一切工作正常，则应该能够看到重复显示的温度、湿度与模拟输入。

模拟输入测试

在 LDR 输入上连接一个 470 Ω 的电阻。观察连续输出的同时，调整 R12，直到其值归 0。这可以证明该部分电路工作正常。接着，针对土壤传感器的 R8 重复这一操作步骤。

DHT22 测试

从 DHT22 输出的数值应该是你所期望的，即当地环境的温度与湿度。你可以使用热风源（比如电吹风）向 DHT22 吹暖风（不要太热！），并观察其反应。

软件功能测试

现在上传完整版本的 GreenShield 软件。测试时，你可以使用 Arduino IDE 的串口监视器窗口。这不是一个测试扩展套件，之前在测试原型时我们已经用过其中一些。

加载好完整的 GreenShield 软件之后，首次启动时你应该能够看到 OK 字样。GreenShield 不提供命令提示符。输入命令 `RY:0:0?` 后，应该得到响应 `RY:0:0`。接着输入 `RY:0:1` 命令，继电器应该发出“咔哒”声，相应 LED 灯应该亮起。输入 `RY:0:?` 命令将返回 `RY:0:1`。

此外，还应该测试其他命令。通过短路或打开模拟输入，可以测试模拟限制。DHT22 是独立的，我们不需要做太多，但你可以把限制范围设置得很小，并用一个热风源与一壶沸水的蒸汽验证设置的温度与湿度限制是否能够如期望的那样工作。

测试坦白

组装第一个 GreenShield 板子时，我没有找到之前购买的松乐 5 V 继电器。我到处寻找但一无所获，最后从元件盒里拿出几个蓝色继电器，并把它们焊在 PCB 上。这些继电器的外形、颜色、类型都对，但就是不响应软件命令。我对此百思不得其解，直到看到印在两个继电器上的文字才意识到自己做了什么。原来是线圈电压不对，它们是 12 V DC 而不是 5 V DC。在此期间，我找到了丢失的 5 V 继电器（其实它们就在我原来放的地方）。不久后，我拆掉了 12 V 的继电器，装上了 5 V 继电器并进行测试。结果显示，继电器的控制命令与功能映射都如预期那样正常工作。

这件事告诫我们：把元件焊接到 PCB 之前，一定要先检查元件规格是否合乎要求。

10.3.8　运行

从物理结构看，GreenShield 扩展板十分简单；但从如何集成到所期望的环境看，其功能又很复杂。GreenShield 有两个主要的可变输入，分别为光线照度与土壤水分，必须应用一组特定条件对它们进行校准。光敏电阻与土壤水分探针的感知范围决定着如何使用微调电位器设置中心值。并非所有 LDR 都一样，有的水分探头只是一对探头，有的还带有一个晶体管，它们之间存在差异。

有两种基本方法可以校正 GreenShield 扩展板：（1）使用某种参考基准，将 GreenShield 校准到已知范围；（2）使用主观评估（比如主观感觉土壤是否足够湿？），根据特定环境调整 GreenShield。

使用校准方法将允许你根据硬数据指定范围。比如，如果知道适于西红柿生长的最合适的土壤含水量，那么进行校准以及计算 ADC 读数如何与含水量进行对应时，你可以使用这些数值。虽然 10.3.4 节中的“原型测试”部分已经介绍了校准 GreenShield 的基本步骤（土壤水分与光线照度），但你仍然可以使用昂贵的实验设备作为数据参考源。

相比较而言，主观方法应用起来更容易些。由于主要目的都是防止植物死亡，所以略微调整之后大概也能得到同样有效的结果。你可以调整微调电位器更改主观评估的上下限，以此获得一个可接受的范围。

在此过程中，我建议大家先做一些试验，尝试一下哪种方法更适合你的实际应用。同时，我也建议你花一些时间记录来自 GreenShield 的数据，并建立一些资料，以便研究如何对实际情况做出最恰当的响应。

10.3.9　后续步骤

GreenShield 扩展板的确很小，但很有潜力。你可以将其与蓝牙扩展板甚至以太网扩展板一起使用，远程监控植物生长的土壤条件，这些植物可以是你最爱的盆栽植物、温室中的西红柿或兰花、户外菜地中的作物、“火星殖民地”的花园。把 24 VAC 的喷洒设备接入其中一个继电器，可以实现自动灌溉功能。你也可以使用另一个继电器启动温室中的通风机降温，或者连接加热器让温室在冬天保持温暖。当然，你也可以添加其他继电器或者带有继电器模块的扩展板（比如第 9 章介绍的那些），进一步增强 GreenShield 的能力。

如果你打算把 GreenShield 绑在森林中的某棵树上，用以采集长期数据，那么可以将 microSD 扩展板与 GreenShield 扩展板一起使用（最好再添加一个太阳能电板，为它供电以保证能够持续工作）。通过与 Wi-Fi 或 GSM 扩展板一起使用，并在大农场的各处安装多个 GreenShield，可以实现对整个农场土壤条件的监测。

> GreenShield 软件目前还无法存储设定点的值，如果通向 Arduino 的电源中断，这些值将会丢失。解决该问题的方法之一是，使用 AVR IC 中的 EEPROM。有关内容请参考第 7 章关于 EEPROM 库的概述。

10.4　制作与Arduino兼容的PCB

制作与 Arduino 硬件兼容的 PCB 非常简单。事实上，Arduino 官方甚至提供电路图与 PCB 布局设计，供大家下载使用。然而，他们并未提供顶部或底部丝印层，因为这部分版权归 Arduino.cc 所有，并且未包含在开放许可证之下。你需要自己为电路板制作图样。

尽管制作一个 Arduino 或扩展板副本很容易，但从经济角度看，克隆现有 Arduino 设计（扩展板或 MCU 板）并不实惠。除非你拥有或有权使用 PCB 产品设备，并且需要成百上千的板子，或者需要某种特定类型的板子。否则你可能会找到一些已经组装好的成品，它们的价格远比你自己制作要便宜得多。

如果现有 PCB 的大小、外形无法满足需要，那么自己动手制作 PCB 的意义就凸显出来了。你可能想把 Arduino 集成到一个更大的系统中，或者把 AVR MCU 放入一个受约束的特殊位置，比如无人机或机器人。如果真是这样，你或许应该考虑制作一个软件兼容的 PCB。你的板子不必非得看起来像 Arduino 一样，也没有必要与现有的扩展板保持兼容。当然，如果你想把它与其他来源的扩展板一起使用，就要考虑相互兼容的问题。

正如第 1 章中提到的，一款设备可以与 Arduino 是软件兼容的，而不必硬件兼容。唯一需要的是一个合适的 AVR 处理器、BootLoader 固件、Arduino 运行时库，甚至 BootLoader 固件也不是必需的，而是可选的。本部分将设计并制作一款基于 AVR 的 DC 电源控制器，适合与重负荷继电器、大功率 LED、AC 或 DC 发动机一起使用。

为你的板子编程

如果你想在一个全新的 AVR 微控制器上使用 ArduinoBootLoader，就必须先把 BootLoader 安装到芯片的内置闪存中。完成此操作之后，你就可以像使用 Arduino 一样使用自己的板子。有关将可执行代码上传到 AVR MCU 的工具与方法，请参考 6.5 节。除了自己动手安装 BootLoader 固件之外，许多供应商销售已经装好 ArduinoBootLoader 的 AVR 设备。你可以在 Adafruit（https://www.adafruit.com）与 SparkFun（https://www.sparkfun.com）上购买预装有 ArduinoBootLoader 的 ATmega328 IC，也可以在 Amazon.com 或 eBay 的搜索框中输入 ATmega328 with Arduino bootloader，将得到大量商品列表。

一旦安装好 BootLoader，你就可以使用 USB 转串口适配器或带有合适 RS-232 模块的标准串口（如图 10-26 所示），或者类似 SparkFun 的 USB 转串行接口板（如图 6-8 所示）。对于某些项目，比如接下来要讲的 Switchinator，如果串口已经处于使用状态，或者没有使用 BootLoader，那么为 MCU 编程时，串口就是必需的。这意味着你将需要使用 ISP（ICSP）编程器，比如 Atmel-ICE 或 USBtinyISP，相关介绍请参考第 6 章。

10.5 Switchinator

下面要介绍的是一个我称之为 Switchinator 的设备，它是一个 14 通道的 DC 开关，带有 4 个模拟输入通道，支持远程控制。最初版本采用 RS-232 接口与简单的命令响应协议，后续版本可能会采用 RS-485 接口以取代 RS-232 接口。

10.5.1 定义与规划

Switchinator 是一个独立的 PCB，使用普通的 ATmega328 或者 ATmega328P MCU，MCU 不带有 ArduinoBootLoader。Switchinator 也可以使用安装 BootLoader 固件的 MCU，MCU 带有串口编程器或 USB 转串口转换器。我选择使用 Arduino IDE 进行编译，使用 Adafruit USBtinyISP 编程。有关为 MCU 编程的更多内容，请参考第 6 章。

核心硬件：

- ATmega328
- 16 MHz 晶体时钟源
- ICSP 编程接口
- 集成 5 V DC 电源供电（9 V~12 V DC 输入）

输入与输出：

- 4 模拟输入
- 14 离散数字输出

控制接口：

- RS-232 主机接口
- 命令响应协议，控制主机驱动
- 按需模拟读数

- 控制主机输出覆写（output override）

Switchinator 提供 14 路离散数字输出与 4 路模拟输入，通过 ICSPSPI 接口针脚阵列，可以使用 SPI 接口。离散数字输出与模拟输入在 PCB 边缘，使用的是螺丝端子台。RS-232 接口用来在板子与控制主机之间进行通信，占用 MCU 的 D0 与 D1 引脚。

该 PCB 完全采用过孔设计。这有利于简化组装，但要求 PCB 更大，并且增加了 PCB 布局的难度。PCB 大小不会超过 100 mm × 140 mm，安装孔位于 PCB 的 4 个边角。

Switchinator 的离散数字输出可以用来驱动继电器，比如 GreenShield 扩展板上使用的那些继电器，或者控制单极步进发动机（最多 3 个），使用的基本电路如图 10-18 所示。

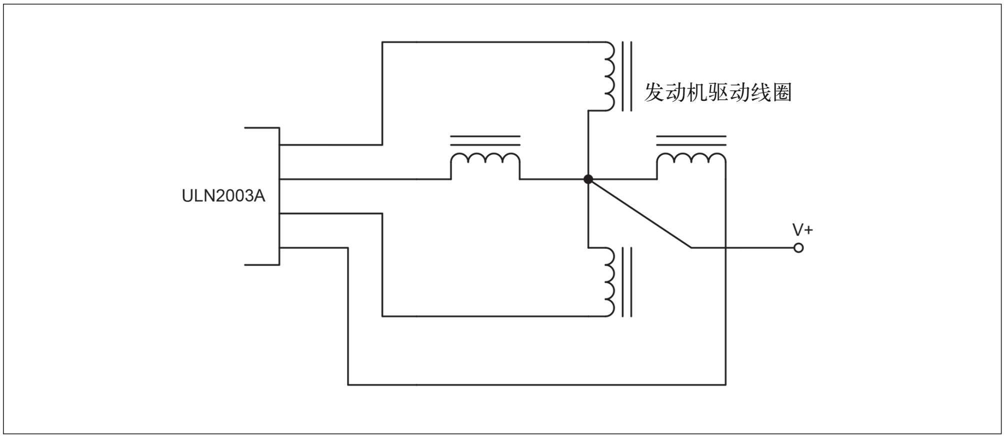


图 10-18：ULN2003A 用作步进驱动器

Switchinator 的数字输出也可以用于驱动大电流 LED、螺线管或 DC 发动机，如图 10-19 所示。

我们将使用基于免焊面包板的基本原型，以开发软件的最初版本。软件的最终版本将在最后的硬件上开发完成。

进行最终硬件设计时，我将使用 Fritzing 制作电路图并设计 PCB 布局。

10.5.2　设计

Switchinator 是单面板，大小为 124 mm × 96 mm，带有 4 个安装孔。最终板子的大小会比采用表面贴装设计的板子更大（也超过了 Eagle 免费版的限制大小，前面制作 GreenShield 板时使用过 Eagle 软件）。

Switchinator 没有外壳与电力供应，这极大地简化了设计。它是完全独立的，运行时只需要一个外部 DC 电源。

1. 功能

Switchinator 是数字输出设备，拥有模拟输入能力。其主要用途是开关 DC 负载，既可以是电感应的（inductive），也可以是无感的（noninductive）。ATmega328 AVR MCU 对输入命令进行解码，并把状态数据返回给控制主机。

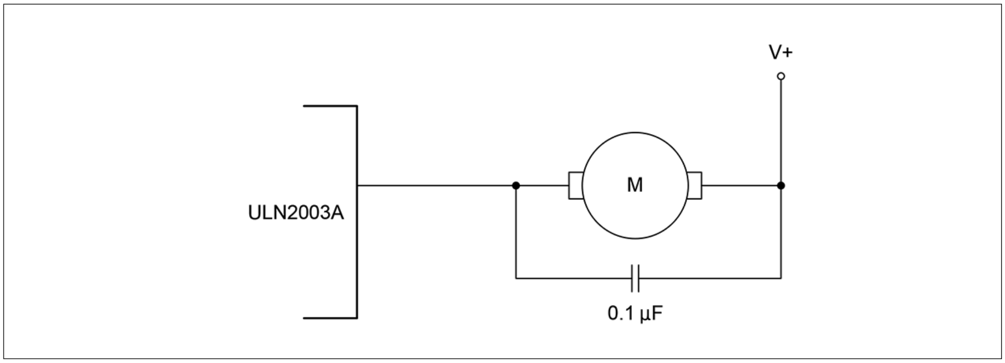


图 10-19：ULN2003A DC 电机驱动器

ATmega328 主要用作命令译码器，把 PCB 上的 I/O 连接到主机系统。虽然它被编程用作 I/O 设备，但也可以被编程基于模拟输入执行自治功能（autonomous functions）。通过使用线性温度传感器（比如 LM35），Switchinator 能够被轻松编程为一个控制器，控制环境试验室或环氧树脂固化室。

Switchinator 由 3 个主要部分组成，分别为 MCU、数字 I/O、电源供应。图 10-20 显示的是 Switchinator 的框图。

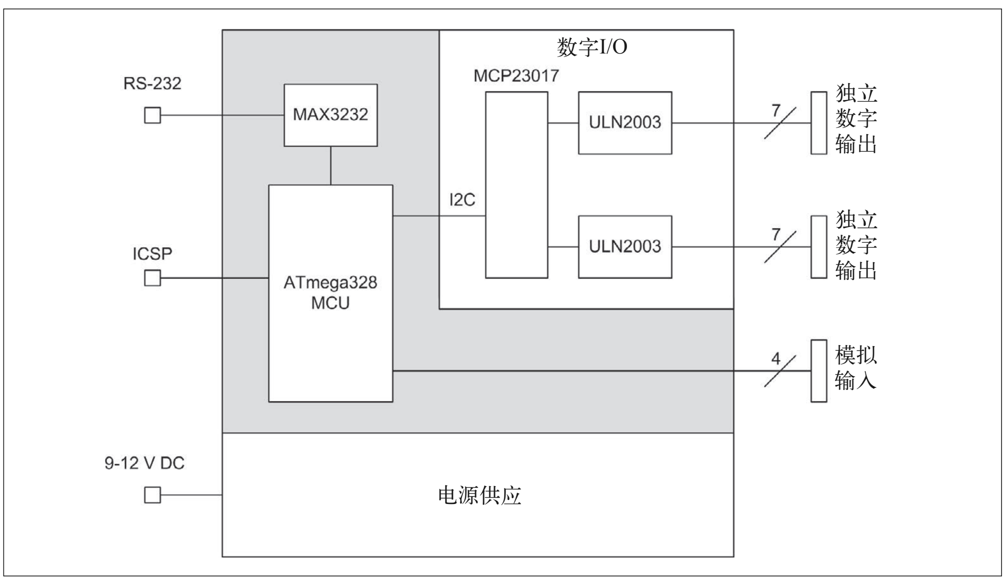


图 10-20：Switchinator 框图

离散数字输出由 ULN2003A IC 驱动。此外还提供 4 个模拟输入，模拟参考电压与 AVCC 电压可以由板载内部电源提供，也可以由外部电源提供。你可以使用跳线选择模拟电压源。

一种简单、易于理解的命令响应协议用来监视与控制 Switchinator。

2. 硬件

Switchinator 拥有 14 路数字输出，每路输出都连接到一对 ULN2003A 驱动器 IC 的达林顿输出。ULN2003A 驱动器每个 IC 拥有 7 个通道，每个 ULN2003A 通道可以处理的电流高达 300 mA 或者更高（某些情况下）。

微芯片 MCP23017 I2C 数字 I/O 扩展板用于驱动 ULN2003 部件，主要为了防止用完 AVR MCU 所有可用的数字 I/O 引脚。AVR 上的两个数字 I/O 引脚用作串口，两个模拟引脚用作连接到 MCP23017 IC 的 I2C 接口。那些未连接的数字 I/O 引脚未被使用，它们可以用在以后的扩展中。

RS-232 接口由 MAX232 TTL 转 RS232 转换器 IC 实现。串口与主机系统（充当主控制器）进行通信。主控制器可以是一台 PC、Arduino，或者是某种带有 RS-232 接口的可编程控制器。DB-9 连接器用作串口。请注意，这不是 RS-232 的完整实现，实现的只是 RxD 与 TxD 信号。Switchinator 没有 USB 连接器。

数字输出与模拟输入采用的是 3.5 mm（0.138 in，138 mil）的螺距螺旋式接线端子。此外，两根跳线用来经由接线端子从外部提供模拟 V+ 与模拟参考电压输入。一个标准的 PCB 安装 DC 筒式连接器提供 DC 电源，允许的电源输入范围为 6 V~12 V DC（最好是 9 V）。一个采用 TO-220 封装的 7805 将电压调节为 PCB 使用的 5 V 电压。图 10-21 是使用 Fritzing 制作的电路图。

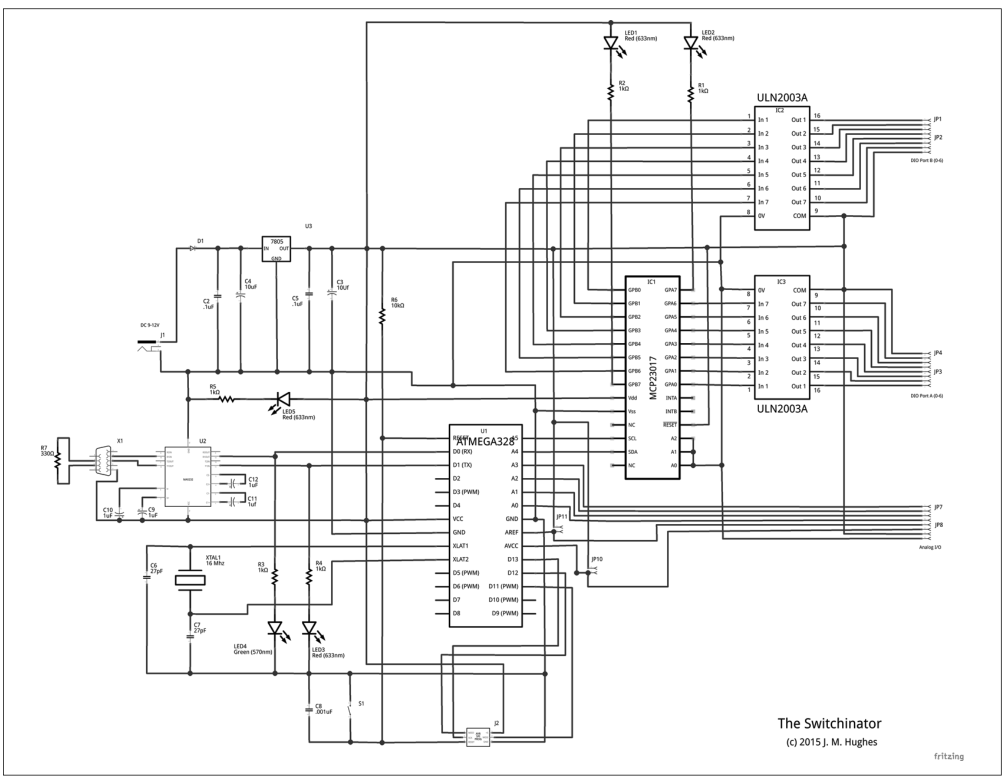


图 10-21：Switchinator 电路图

图 10-21 中使用的电路图符号说明，当工具库中的元件不遵从相同的大小与间距约定时，会发生什么。使用开源工具时经常会遇到这样的问题，因为不是所有元件都遵循相同规则。当然，这样也是可以的，只是让人觉得有些奇怪。

MCP23017 数字 I/O 扩展器

MCP23017 带有 I2C 接口，其“兄弟”MCP23S17 带有 SPI 接口，它们都是寄存器控制的 I/O 设备，在主设备与离散数字输入或输出（最多 16 路）之间传送二进制信号。关于 MCP23017 内部原理的讨论同样适用于 MCP23S17。

MCP23017 与 MCP23S17 通常用在 I/O 扩展板上，比如第 8 章中介绍的那些，用来对 MCU 的 I/O 能力进行扩展。通过地址引脚，可以把多个 MCP23017 IC（多达 8 个）连接到 AVR MCU 上，产生 128 个通道的离散数字 I/O。

MCP23017 的行为由一系列控制与数据寄存器的内容定义。主控设备可以随时通过 I2C 或 SPI 接口读取或设置寄存器的值。图 10-22 显示的是 MCP23017 的框图。MCP23S17 与 MCP23017 完全相同，只是它集成的是 SPI 接口，而非 I2C 接口。

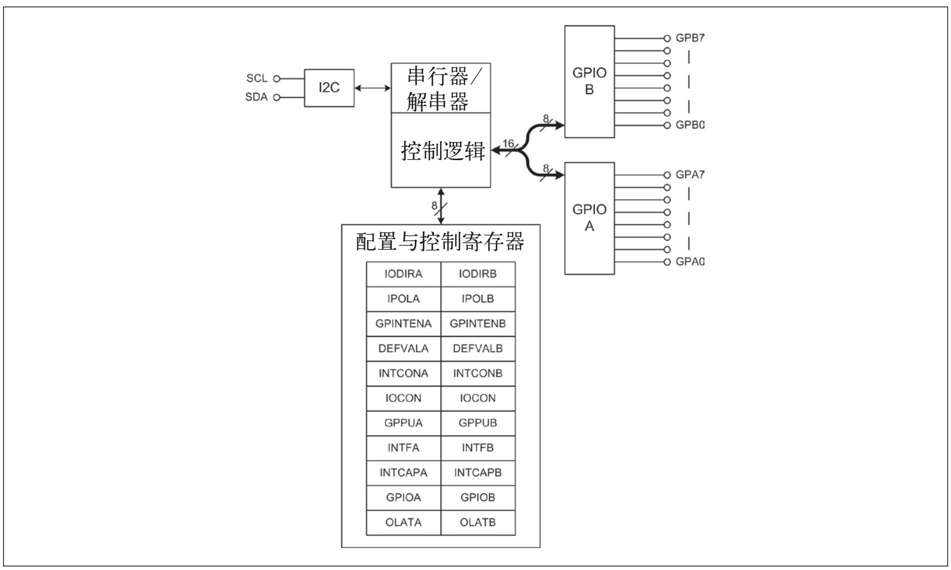


图 10-22：MCP23017 I/O 扩展器

MCP23017 内部有 22 个寄存器，分为 A 与 B 两组，A 组寄存器使用 GPIO A 端口，B 组寄存器使用 GPIO B 端口。端口方向（输入或输出）分别由 IODIRA 与 IODIRB 寄存器设置。端口极性（高电平有效或低电平有效）则分别由 IPOLA 与 IPOLB 寄存器设置。GPINTENA 与 GPINTENB 启用中断发生器。GPPUA 与 GPPUB 启用上拉电阻。端口 A 或端口 B 引脚的信号是从 GPIOA 或 GPIOB 寄存器读取的。OLATA 与 OLATB 寄存器返回内部输出

锁存器的状态，当端口处于输出模式并且有数据写入 GPIOA 或 GPIOB 寄存器时，这些锁存器即被设置。表 10-9 列出了整个寄存器组。请注意，寄存器地址是十六进制形式 (Addr=MCP23017 寄存器空间地址)，POR/RST 对应于上电复位或外部复位。

表10-9：MCP23017控制寄存器（IOCON.BANK = 0）

寄存器	Addr	位7	位6	位5	位4	位3	位2	位1	位0	POR/RST值
IODIRA	00	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0	1111 1111
IODIRB	01	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0	1111 1111
IPOLA	02	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0	0000 0000
IPOLB	03	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0	0000 0000
GPINTENA	04	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2	GPINT1	GPINT0	0000 0000
GPINTENB	05	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2	GPINT1	GPINT0	0000 0000
DEFVALA	06	DEF7	DEF6	DEF5	DEF4	DEF3	DEF2	DEF1	DEF0	0000 0000
DEFVALB	07	DEF7	DEF6	DEF5	DEF4	DEF3	DEF2	DEF1	DEF0	0000 0000
INTCONA	08	IOC7	IOC6	IOC5	IOC4	IOC3	IOC2	IOC1	IOC0	0000 0000
INTCONB	09	IOC7	IOC6	IOC5	IOC4	IOC3	IOC2	IOC1	IOC0	0000 0000
IOCON	0A	BANK	MIRROR	SEQOP	DISSLW	HAEN	ODR	INTPOL	—	0000 0000
IOCON	0B	BANK	MIRROR	SEQOP	DISSLW	HAEN	ODR	INTPOL	—	0000 0000
GPPUA	0C	PU7	PU6	PU5	PU4	PU3	PU2	PU1	PU0	0000 0000
GPPUB	0D	PU7	PU6	PU5	PU4	PU3	PU2	PU1	PU0	0000 0000
INTFA	0E	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INTO	0000 0000
INTFB	0F	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INTO	0000 0000
INTCAPA	10	ICP7	ICP6	ICP5	ICP4	ICP3	ICP2	ICP1	ICP0	0000 0000
INTCAPB	11	ICP7	ICP6	ICP5	ICP4	ICP3	ICP2	ICP1	ICP0	0000 0000
GPIOA	12	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0	0000 0000
GPIOB	13	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0	0000 0000
OLATA	14	OL7	OL6	OL5	OL4	OL3	OL2	OL1	OL0	0000 0000
OLATB	15	OL7	OL6	OL5	OL4	OL3	OL2	OL1	OL0	0000 0000

MCP23017 拥有多种功能，每种功能可能对应于特定的应用场景。对于简单的数字 I/O 应用，只需要读写 GPIO 寄存器就可以了。有关寄存器的更多细节，请参考微芯的 MCP23017/ MCP23S17 数据手册（http://bit.ly/micro-mcp）。

使用 MCP23017 的另一个问题是，如何与它进行通信。两阶段协议（two-phase protocol）用来访问内部寄存器，第一步是发送控制字节以及要访问的寄存器地址，第二步是读寄存器或写寄存器。通过控制字节（寄存器地址字节对）选择新的寄存器之前，最近选择的寄存器处于活动状态。

控制字节是 MCP23017 的 7 位地址，带有读写（R/W）位。在单 IC 系统中，A0、A1、A2 位总为 0，第 6 位预定义为 1。这样一来，如果是写操作，则为 0x20；如果是读数据，则为 0x21（在控制字节中，读写位是最低有效位）。

有一些现成的库可以与 MCP23017 和 MCP23S17 进行交互，自己编写库也并不难。示例 10-7 中的代码展示了如何处理 I/O 定向寄存器，以及如何使用 I2C 接口与 Wire 库将 GPIOA 寄存器作为输入。

示例 10-7　访问 MCP23017 中的 IODIRA 与 GPIOA 寄存器

```
//发送控制与寄存器地址字节
Wire.beginTransmission(0x20);
Wire.write(0x00);
//写数据到IODIRA寄存器
Wire.write(0xFF);
//结束会话
Wire.endTransmission();

//发送控制与寄存器地址字节
Wire.beginTransmission(0x20);
Wire.write(0x12);   // GPIOA
Wire.endTransmission();
//从端口A读数据
Wire.requestFrom(0x20, 1); // 设置R/W位为1(read)
portval = Wire.read();  // 仍然指向0x12
```

由于上电或重置后，GP I/O 端口默认被配置成输入（见表 10-9），所以第一步可以忽略。为了将端口配置成输出，应该把 IODIRA 位设置为 0，如示例 10-8 所示。

示例 10-8　写数据到输出端口

```
//发送控制与寄存器地址字节
Wire.beginTransmission(0x20);
Wire.write(0x00);
//写数据到IODIRA寄存器
Wire.write(0x0);
//结束
Wire.endTransmission();

//发送控制与寄存器地址字节
Wire.beginTransmission(0x20);
```

```
    Wire.write(0x12);   // GPIOA
    //写数据到端口A
    Wire.write(portval);
    Wire.endTransmission();
```

MCP23S17 的 SPI 接口也采用类似的工作方式，若需要，甚至可以使用地址引脚。其他所有行为与 MCP23017 完全一样。不论是 I2C 还是 SPI，其 I/O 运行速度一样快。

在 Switchinator 中，我使用了一个 16 MHz 的晶振作为 MCU 时钟，主要是因为我有一堆 16 MHz 晶振。对于 Switchinator，AVR 内部的 RC 振荡器能工作得很好。尽管晶振允许 MCU 以特定速率运行，但这也会增加内部熔丝位的复杂度，AVR MCU 使用熔丝位做内部配置。有关熔丝位的内容，请参考 10.5.3 节中的“为 16 MHz 晶振设置 AVR MCU 熔丝位”部分。

4 个模拟输入被引出到一对四位的接线端子。其余位置用作 V+、地、可选模拟参考、AVCC 输入。跳线 JP10 与 JP11 用于选择外部电压源，将其从 PCB 移除即可实现。

MCU 的两个数字引脚（D0 与 D1）用作 RS-232 串口，D11、D12、D13 这 3 个数字引脚用作 ICSP 编程接口，其他数字引脚未分配。4 个模拟输入（A0、A1、A2、A3）用作通用模拟输入，引脚 A4 与 A5 用作 MCU 与 MCP23017 之间的 I2C 接口。表 10-10 列出了 Switchinator 的引脚分配情况。

表10-10：Switchinator MCU引脚分配

引脚	用途	引脚	用途	引脚	用途
D0	RxD for RS-232	D7	未使用	A0	通用模拟输入
D1	TxD for RS-232	D8	未使用	A1	通用模拟输入
D2	未使用	D9	未使用	A2	通用模拟输入
D3	为使用	D10	未使用	A3	通用模拟输入
D4	未使用	D11	ICSP MOSI	A4	I2C SCL
D5	未使用	D12	ICSP MISO	A5	I2C SDA

根据图 10-21 的电路图，可以详细列出需要的元件列表，如表 10-11 所示。

表10-11：Switchinator元件列表

数量	元件描述	数量	元件描述
2	电容，0.1 μF	4	红色 LED
2	电容，10 μF	1	绿色 LED
2	电容，27 pF	5	1K Ω 电阻，5%，1/4 W
1	电容，0.001 μF	1	10K Ω 电阻，5%，1/4 W
4	电容，1 μF	1	330 Ω 电阻，5%，1/4 W
1	1N4001 二极管	1	开关，触感按钮
1	MCP23017 I2C I/O 扩展器	1	ATmega328 MCU，28 针双列直插封装
2	ULN2003A 输出驱动器	1	MAX232 RS232 收发器，16 针双列直插封装

（续）

数量	元件描述	数量	元件描述
1	电源插座，5.5 mm 筒式	1	7805 调压器，T0-220
1	AVR ISP 连接器，2 × 3 引脚	1	DB9 公连接器，PCB 安装
6	接线端子，3.5 mm 间距	1	16 MHz 晶振
2	2 针跳线头	1	定制 PCB

3. 软件

与 GreenShield 项目一样，Switchinator 最复杂的部分其实是软件。图 10-23 显示了软件的框图。

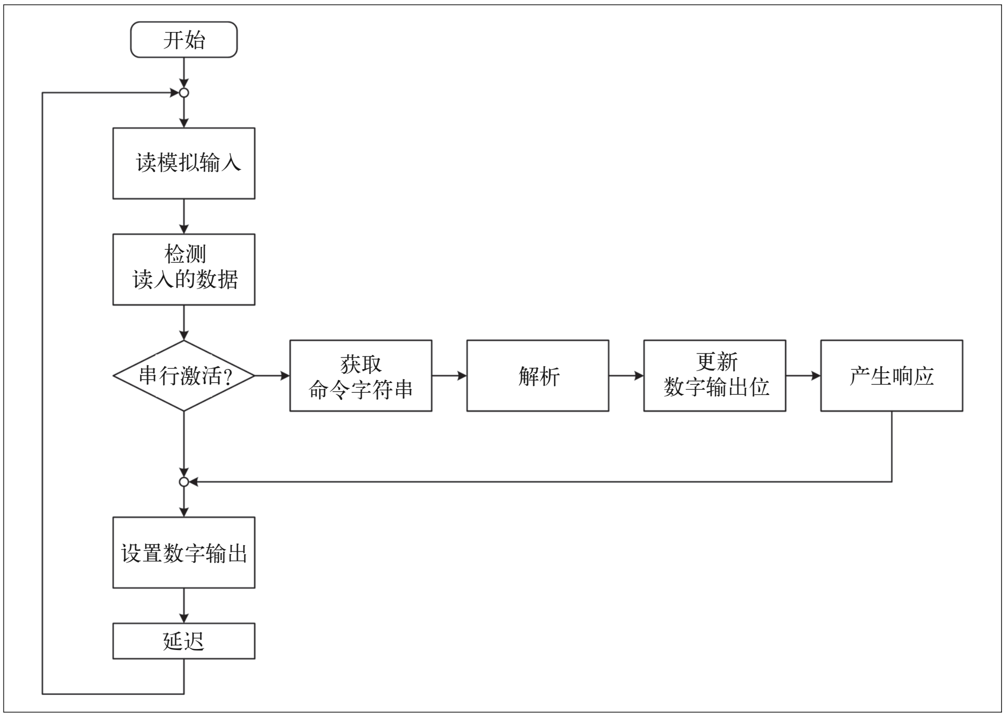


图 10-23：Switchinator 软件框图

Switchinator 等待输入命令时，它会发送一个 > 字符，后面跟着一个空格。响应前面是 < 字符与空格，后面紧跟着响应数据。使用 > 与 < 字符主要是为了方便在控制主机系统中运行软件。所有输入与输出行都以新行字符（\n 或 0x0A）结束。

每次运行主循环都会读取模拟输入，并保存读入的值。要求读取模拟输入时，这些读入值就会被返回给主控机，这意味着它们的更新速度与主循环是一样的。

接下来，软件检查传入的串行数据。如果输入缓冲区中有数据，就会调用输入读取函数不断读取字符，直到遇到新行符号（ASCII 值为 10，0x0A 或 \n）。然后解析器从字符串提取命令，解析参数，执行命令指定的操作。

Switchinator 通过简单的命令响应协议进行控制。对于控制主机发出的每个命令或查询，Switchinator 都会用单个响应进行回复。Switchinator 不会主动发起与控制主机的通信事务。表 10-12 列出了完整的命令响应协议。

表10-12：Switchinator命令响应协议

命令	响应	描述	输入/输出格式
`A:`*n*	`A:`*n*`:`*val*	获取 raw DN 中的模拟输入 *n*	Hex
`R:`*n*`M`	`R:`*n*`:`*val*	读取输出 *n* 的状态	Hex (0 or 1)
`W:`*n*`:`*val*	`OK`	写 0 或 1 到输出 *n*	Hex
`S:`*val*	`OK`	把所有输出设为十六进制值	4 位十六进制
`G:?`	`G:`*val*	获取所有输出的十六进制值	4 位十六进制

如上所列，Switchinator 命令集很简单，其中只有 `W`（写）命令使用两个参数，其他命令均为一个参数。所有参数与响应值均采用十六进制表示法。这允许数字端口号为单个十六进制数，而模拟输入值（`A` 命令）、设置与获取命令（`S` 表示设置所有端口，`G` 表示获取所有端口）使用 4 位十六进制值。数字 0 与 1 名义上是十六进制，但它们在十进制表示法中也是一样的。

检查 Switchinator 返回的字符串，以进行错误检测。如果 `A`、`R`、`G` 命令返回发送的字符串，则表示有错误发生。如果 `S` 或 `W` 命令遇到错误，它们会返回原字符串；如果成功，则返回字符串 OK。

八进制与十六进制数

如果你已经知道如何使用八进制与十六进制表示法，那么可以跳过本部分内容。否则，让我们一起简单地回顾历史，了解这些数制从何而来，以及它们是如何工作的。

早期的计算机体积庞大，能够塞满整个房间，并且产生大量热量，足以烤熟你早餐中的即食燕麦片。很快人们就认识到，处理二进制数的效果不佳。并且，由于机器内部采用的是二进制，十进制数值无法优雅地映射到位模式（显示在控制面板指示灯上），以及在内存中进行保存。

针对这一问题的解决方法是，使用其他数制（非二进制或十进制）的数字来表示数值。最后发现有两种数制系统——八进制（base-8）与十六进制（base-16）——可以直接映射到二进制值。

八进制数字系统在使用 12 位、24 位、36 位数据字与寻址的计算机中很流行，因为每个粒度值（size value）都能被 3 整除。八进制数值为 0~7，因此每个八进制数对应 3 位二进制数。表 10-13 显示了八进制如何工作。

在现代 Unix 与 Linux 计算机中仍然能看到八进制数，用作文件权限位，此外几乎看不到它们的身影。现在，人们已经不再使用 12 位或 24 位的通用计算机，但一些研究小组仍然在为特定应用设计 24 位的微处理器。Freescale 生产的 DSP56303 DSP（数字信号处理器）可以作为 24 位机器使用。

表10-13：12位计算机的八进制系统

十进制	二进制	八进制	十进制	二进制	八进制
0	000 000 000 000	0000	9	000 000 001 001	0011
1	000 000 000 001	0001	10	000 000 001 010	0012
2	000 000 000 010	0002	20	000 000 010 100	0024
3	000 000 000 011	0003	30	000 000 011 110	0036
4	000 000 000 100	0004	40	000 000 101 000	0050
5	000 000 000 101	0005	50	000 000 110 010	0062
6	000 000 000 110	0006	100	000 001 100 100	0144
7	000 000 000 111	0007	200	000 011 001 000	0310
8	000 000 001 000	0020	511	000 111 111 111	0777

随着计算机体系结构的变化，数据与地址的大小变为 4 的倍数，此时八进制系统就变得很麻烦。解决方案是使用十六进制系统。在十六进制系统中，每位数为 0~15 的值，用二进制表示为 0000~1111。换言之，每个十六进制数是 0.5 B，两个十六进制数表示 1 B（8 位）。

然而，使用十六进制表示法会遇到一个问题，那就是不能用一位数表示任何大于 9 的十进制数值，比如数值 10 要写成两位数形式。在 20 世纪 50 年代，有人提出了不同的解决方案，试图解决数制表示问题，其中的某些方案以现在的眼光看让人觉得相当奇怪。

几年之后，终于尘埃落定，人们最终使用 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F 表示 0~15 的数。表 10-14 显示了十进制、二进制、十六进制之间的关系。

表10-14：16位计算机的十六进制记数系统

十进制	二进制	十六进制	十进制	二进制	十六进制
0	0000 0000 0000 0000	0000	10	0000 0000 0000 1010	000A
1	0000 0000 0000 0001	0001	20	0000 0000 0001 0100	0014
2	0000 0000 0000 0010	0002	30	0000 0000 0001 1110	001E
3	0000 0000 0000 0011	0003	40	0000 0000 0010 1000	0028
4	0000 0000 0000 0100	0004	50	0000 0000 0011 0010	0032
5	0000 0000 0000 0101	0005	100	0000 0000 0110 0100	0064
6	0000 0000 0000 0110	0006	200	0000 0000 1100 1000	00C8
7	0000 0000 0000 0111	0007	255	0000 0000 1111 1111	00FF
8	0000 0000 0000 1000	0008	511	0000 0001 1111 1111	01FF
9	0000 0000 0000 1001	0009	4095	0000 1111 1111 1111	0FFF

有时，你会在软件中看到一些十六进制数值，比如在 C 与 C++ 语言中写作 0x3F，在汇编语言中写作 3Fh，在 Forth 语言中写作 $3F，在 URL 中写作 %3F。通过调整某些数字的显示，使用七段数字 LED 显示器可以显示十六进制数，把 A、b、c、d、E、F 显示出来，如图 10-24 所示。

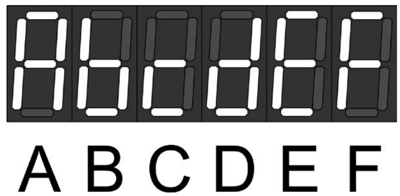


图 10-24：用传统的 7 段 LED 显示器显示十六进制数

很快人们就发现，八进制与十六进制记数系统中有一些特殊的值——有时也称为幻数（magic numbers）——不断出现。比如十六进制中，0xFF 的所有位都是 1。0x5A5A 是一种交替模式（二进制数为 0101 1010 0101 1010），有时用作填充检查栈的使用情况；或者用于重写磁盘驱动器，把之前存储的数据模糊掉。0x7F 是 127，0x1FF 是 511，0x3FF 是 1023。0x3FF（1023）是 10 位 ADC（ATmega328 AVR）能产生的最大值，0x1FF 是中心值。

做数学运算时，既可以使用八进制数，也可以使用十进制数。在汇编语言编程中，经常需要对十六进制的地址值做加减运算（比如间接跳转或相对转移指令）。当然，使用十六进制数做乘除运算也可以，但这确实需要做一些练习才能掌握。除非你想或者需要做大量低级汇编语言编程工作，否则完全没有必要学习有关十六进制运算的内容。但为了使用微控制器，你应该能够将十六进制数逐位转换为二进制数（或者反之）。

`A` 命令接收 0~3 的单位数字，模拟数据是 AVR MCU 的 ADC 返回的真实值。它返回的是 1~3 位的十六进制值，0x3FF 是可能返回的最大值。

`R` 与 `W` 命令有特定数字输出端口号，形式为单个十六进制数，范围是 0~0xF（15）。输出通道 7 与 0xF 供板载 LED 使用，它们未通过 ULN2003A 驱动器引出。

请注意，表 10-12 中，为了控制或获取一个以上输出的状态，`S` 与 `G` 命令要使用十六进制值。通过这种方法，我们可以启用（设置为 on 状态）非邻接输出，或者读回所有输出的状态。

比如，如果将输出 5、6、12、13 设置为 on 状态，那么应该发送 `3060h`，转换成二进制如下：

```
位        | 15   11    7    3  0
-----------+----+----+----+---
二进制  |  0011 0000 0110 0000
十六进制|     3    0    6    0
```

请记住，通道编号从 0 开始。

发送十六进制值将引起输出改变状态，以便匹配命令值。这意味着，如果输出处于 on 状态，并且命令值在对应位置为 0，则输出将被设置为 off。

可以通过“读 - 修改 - 写”操作（获取位，修改数据，设置位）设置或清除特定的输出位，而不会干扰其他位。这正是 `W` 命令的工作原理。

循环中的最后一步是，将数字输出状态位传送到硬件。如果从上次更新以来没有发生改变，则输出什么也不做。否则，发生改变的位就表现为 ULN2003A 输出的 on 或 off 状态的改变。

10.5.3 原型

制作原型主要关注 RS-232 接口与命令响应控制协议，原型硬件主要由安装在免焊面包板上的 ATmeta328 组成。RS-232 适配器模块用作 MAX232（在最终产品中使用）的“替身”。图 10-25 显示的是搭建好的原型。

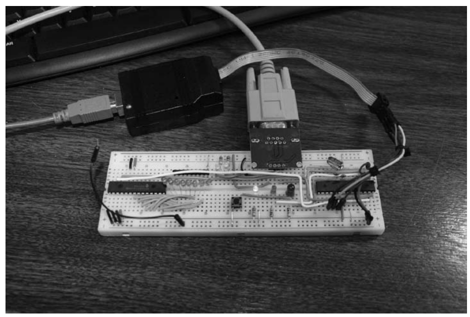


图 10-25：搭建好的 Switchinator 原型

RS-232 模块是自包含单元（self-contained unit），带有 MAX3232 IC，它在功能上等同于 Switchinator 中使用的 MAX232。名义上 MAX3232 是一个 3.3 V 的元件，不过它可以兼容 5 V 电压。此外，它在价格上也比 MAX232 稍贵一些。图 10-26 显示的是 RS-232 接口模块的特写。

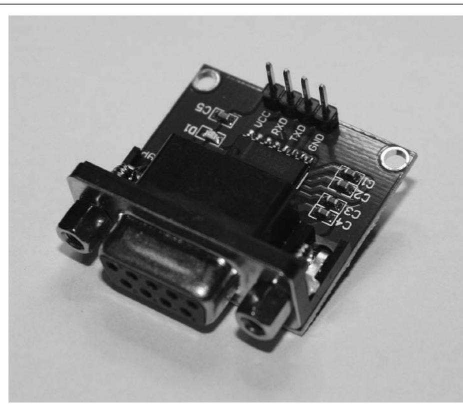


图 10-26：RS-232 接口模块

你可以从多家供应商那里购买到这些模块，价格为 3~4 美元。在谷歌搜索中输入 arduino rs232 module 或 arduino rs232 converter，将会看到大量搜索结果。

大多数台式电脑仍然带有一个 RS-232 端口和一个 DB-9 连接器（位于背面板）。但在你的新型笔记本电脑上，你将找不到可用的串行接口。为了解决这一问题，需要使用 USB 转 RS232 适配器，价格为 4~30 美元，某些专业类型价格会更高一些。关于 RS-232 与匹配连接器的更多详细内容，推荐你阅读我的另外两本书 *Real World Instrumentation with Python* 和《电子工程师必读：元器件与技术》（见附录 D）。两本书都讲解了有关 RS-232、DB-9 连接器、公母接头转换器的内容，以及如何连接 DB-9 在无交握信号的条件下实现 RxD/TxD 通信。

为 16 MHz 晶振设置 AVR MCU 熔丝位

不同于本书的其他示例，Switchinator 不是 Arduino 板。安装 BootLoader 后，Switchinator 与 Arduino 软件兼容，但安装 BootLoader 不是必需的。本质上，Switchinator 只是一种 AVR MCU 设计。

这意味着 MCU 未进行预先设定，这点与 Arduino 板上的 MCU 不同。Atmel 新推出的 MCU 运行在 RC 时钟模式之下，使用频率约 8 MHz 的内部振荡器。如果安装了 BootLoader，那么某些内部开关设置会指示一部分闪存空间已被保留。一般来说，带有预载 BootLoader 固件的 MCU 也会为外部 16 MHz 的晶振提供熔丝位设置，但全新部件只拥有默认配置。

为了配置 MCU，使其可以使用 16 MHz 晶振作为时钟源，需要对熔丝位进行设置。关于 AVR MCU 熔丝位设置的更多细节，请参考 3.4 节。当然，信息的确切来源是你所用 AVR MCU 的 Atmel 数据手册。

如果想使用 AVRDUDE 的 Arduino 配置（并且拥有 Linux 系统），可以使用如下命令为 ATmega328 做这项工作：

```
sudo avrdude -cusbtiny -p atmega328 -U lfuse:w:0xFF:m \
-U hfuse:w:0xDE:m -U efuse:w:0x05:m
```

如果是 ATmega328p，则需要稍微修改元件参数，如下所示：

```
sudo avrdude -cusbtiny -p atmega328p -U lfuse:w:0xFF:m \
-U hfuse:w:0xDE:m -U efuse:w:0x05:m
```

在 Linux 系统中，如果为 USB I/O 设置了权限规则，那么不必使用 `sudo` 运行 `avrdude`。关于命令行选项以及 `avrdude` 支持的交互命令的更多信息，请参考 avrdude 用户手册（http://bit.ly/avrdude-manual）。

最后一步是，通知编译器目前正在以 16 MHz 运行。为此需要定义 `F_CPU`，如下所示：

```
#define F_CPU 16000000UL
```

使用传统的 Arduino 板时，上面这行宏定义通常由 Arduino 开发环境添加。但使用自定义目标板时，可能需要明确为其指定时钟频率。

Windows 与 Mac OS 版本的 AVRDUDE 的行为方式与 Linux 完全一样。关于 Windows 版本的 AVRDUDE 的更多信息，请参考相关教程（http://www.ladyada.net/learn/avr/setup-win.html）。

1. 原型软件

原型软件基本上与最终版本软件是一样的，只是没有通过 MCP23017 设置数字输出的代码。原型软件的重点是实现命令响应协议。输出状态使用软件中 16 位字的位来表示。

10.5.4 节对此有更详细的讨论，所以此处不会讲解细节内容，以免与后面重复。软件使用 Arduino IDE 进行编译，然后使用 Adafruit USBtinyISP ICSP 接口设备上传到 AVR MCU。

我使用 Arduino IDE 做编译，但禁用了其“首选项”对话框中指定的内部编辑器。这样就可以使用一个不同的编译器（我以写商业与科学软件为生，所以对文本编辑器有些偏爱——我不怎么喜欢 Arduino IDE 的编辑器）。在“工具”菜单中，把板子设置为 Duemilanove with ATmega328，将编程器设置为 USBtinyISP。

在 Linux 系统下，USBtinyISP 不使用虚拟串口（pseudo-serial port），而是直接与下层的 USB I/O 子系统进行通信。尝试以编程者的身份运行 Arduino IDE 将会导致权限错误。可以使用 `sudo` 运行 Arduino IDE，但对于传送代码而言，这种方式非常不便。为了解决这个问题，你需要为 `udev` 添加一个访问规则。

在 /etc/udev/rules.d 中创建一个文件，命名为 tinyusb.rules，然后在其中添加如下字符串：

```
SUBSYSTEM=="usb", ATTR{product}=="USBtiny", ATTR{idProduct}=="0c9f", \
ATTRS{idVendor}=="1781", MODE="0660", GROUP="dialout"
```

我使用 `vi` 与 `sudo` 完成这项工作：

```
sudo vi tinyusb.rules
```

然后重启 `udev` 子系统，使新规则生效。

```
sudo restart udev
```

当然，也可以使用其他类型的编程设备（甚至是另一个 Arduino，请参考第 6 章）。我有一个 Atmel-ICE 工具，但不在 Linux 下使用它，因为我没有最新版本的 AVRDUDE，并且我也不愿意用配置文件做编译调整。Atmel-ICE 支持 Atmel AVR Studio 软件，所以如果你使用的是 Windows 系统，可以尝试使用它。在我的 Linux 系统下，Aadfruit 的小工具工作得很好。

我使用的 IDE 中，点击上传图标后，IDE 将尝试通过 Arduino 板内置的 USB 转串口方法进行传送（由 USB 转串口转换器提供支持，比如之前提到的那个）。这并不适用于 USBtinyISP，所以我使用“文件”菜单下的“使用编程器上传”菜单选项。

我注意到，AVRDUDE 传输软件开始时运行得很慢，但一旦与 AVR MCU 建立可靠的通信连接之后，它就会运行得很快。无论在 Arduino IDE 中，还是从命令行运行 AVRDUDE，你都将看到这个现象。如果遇到错误，请不要惊慌。出现问题时，AVRDUDE 最终会超时，并且告诉你出错位置。

2. 原型测试

原型测试很简单，本部分讲解的测试主要针对命令解析器。如果能够与软件顺利进行交互，则表明 RS-232 接口部分的代码工作正常。此处假设外部 RS-232 收发器能够像预期的那样进行工作。

首先，测试输出（`OUT`）、状态（`ST`）、输入（`AN`）命令。`OUT:A:1` 与 `OUT:A:0` 命令将所有输出设置为 on 或 off。每个输出状态都会被保存到内存，所以此时并不需要输出硬件。

在 `ST:`*`n`*`:?` 命令中，参数 *`n`* 是 0~13 之间的单数位（十六进制中的 D）。请注意，`OUT:`*`n`*`:0` 与 `OUT:`*`n`*`:1` 命令中也使用了单数位（single digit）。如果软件工作正常，则应该能把所有输出设为 off（`0`），然后有选择地启用和禁用 0~13 的任意一个输出，同时不会改变其他任何输出。

接下来，测试模拟输入（`AN`）命令，具体做法是向 A0~A3 应用可变电压源（仅限 0 V~5 V），并请求获取它们的值。输入电压发生变化时，返回的数据也应该相应地发生变化。输入值以 3 位十六进制值的形式返回，其中两个高位字节总为 0（AVR MCU 只有一个 10 位 ADC）。

如前所述，`SP:val` 与 `GP:?` 命令使用 4 位十六进制值。测试时将把所有奇数输出设置为 on，把所有偶数输出设置为 off，然后使用 `ST` 命令检查每个输出的状态。之后，再把所有奇数输出设置为 off，把偶数输出设置为 on，再使用 `ST` 命令检查输出位的状态。

10.5.4　软件

为搭建好的原型而创建简化版软件通常是可行的，也是明智的。就 Switchinator 而言，其软件由多个代码文件组成，其中主文件是 Switchinator.ino，其他文件包括全局宏定义文件、全局变量声明文件、命令解析器文件、响应发生器、I/O 控制代码文件。在原型版本软件中，I/O 模块（sw_io.cpp）不是必需的。

1. 源代码组织

Switchinator 源代码由 8 个文件或模块组成，如表 10-15 所示。其中，主模块为 Switchinator.ino，包含 `setup()` 与 `loop()` 两个函数，并且通过 `#include` 语句引用了其他模块。

表10-15：Switchinator源代码模块

模块	功能
Switchinator.ino	软件主模块，包含 `setup()` 与 `loop()` 两个函数
sw_defs.h	常量定义（`#define` 语句）
sw_gv.cpp	全局变量
sw_io.cpp	包含文件
sw_parse.cpp	命令解析
sw_parse.h	包含文件

对于上表列出的 Switchinator 所有源文件，你都可以从 GitHub（https://github.com/ArdNut/Switchinator）下载。

2. 软件描述

与其他任何一个 Arduino 程序一样，`Switchinator` 软件的行为开始于主程序文件 `Switchinator.ino`，其中包含的代码如示例 10-9 所示。

示例 10-9　Switchinator.ino

```
//------------------------------------------------------------------
// Switchinator.ino
```

```
//
// 创建于2016年,作者J.M.Hughes
// 第10章
//-------------------------------------------------------------------

#include <stdint.h>
#include <Wire.h>

// 包含所有源模块
#include <IOexp.h>

// 定义时钟频率
#include "sw_parse.h"
#include "sw_gv.h"
#include "sw_defs.h"
#include "sw_io.h"

// 启动信息与开始标记
#define F_CPU 16000000UL

bool waitinput = false;
bool usedelay = false;

void setup()
{
    Serial.begin(9600);

    // 开始标记
    Serial.println();
    Serial.println("SWITCHINATOR V1.0");
    Serial.println("READY");
    Serial.println("####");      // Start flag
}

void loop()
{
    // 读取模拟输入与更新GV数组
    ScanAnalog();

    // 发出输入提示
    if (!waitinput) {
        waitinput = true;
        Serial.print(INPPCH);
    }

    // 检查输入命令
    if (GetCommand()) {
        waitinput = false;      // 重置输入提示输出标记
    }
    else {
        ClearBuff(0);
        ResetBuffLen();
        // 若无输入,则启用循环延迟
        usedelay = true;
    }
```

```
        // 解析命令(若接收到)
        if (DecodeCommand()) {
            // 向主机(或用户)返回响应
            Serial.println();
            Serial.print(OUTPCH);
            Serial.println(gv_cmdinstr);
        }

        // 使用状态位更新数字输出
        SetDigBits();

        // 若未检测到输入,则简单地延迟
        if (usedelay) {
            usedelay = false;
            delay(50);
        }
    }
```

上述代码中，`setup()` 函数包含常见的初始化语句，还有初始启动信息。`loop()` 函数不断更新输出位，并通过调用 SW_parse.cpp 模块中的 `GetCommand()` 函数检测来自主控机系统的命令。

从图 10-23 可以看到，“获取命令字符串”（Get command string）对应的就是 sw_parse.cpp 模块中的 `GetCommand()` 函数，而“解析”（Parse）与“产生响应”（Generate response）功能包含于 `DecodeCommand()` 函数，“更新数字输出位”则由 sw_io.cpp 模块中的 `SetDigBits()` 函数负责。图 10-27 描述了如何使用输出位状态缓冲（`gv_statebits`）保存输出位的内部表示。所有位修改操作都基于 `gv_statebits`，而非实际的数字输出。

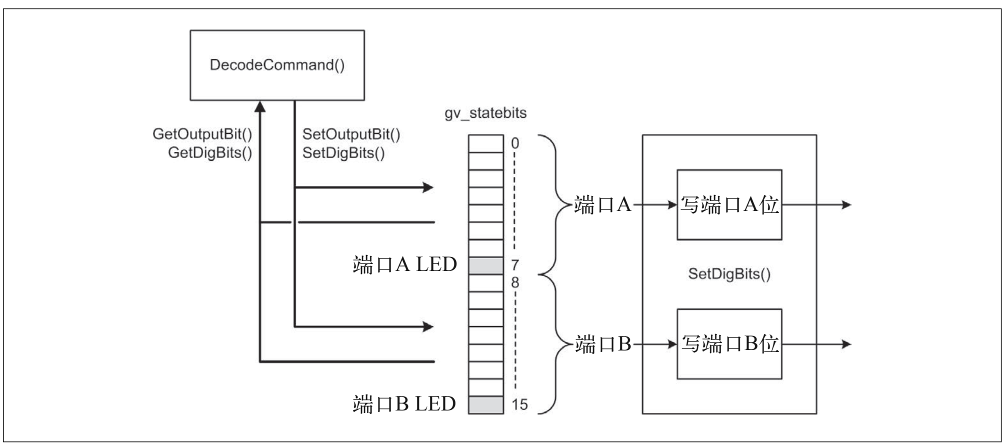


图 10-27：操作中的 Switchinator 虚拟位缓冲

请注意，向端口 A 或端口 B 写 0x00 也会开启相应端口的 LED 灯，而向端口 A 或 B 写入 0xFF 则会禁用 LED 灯。如果不再有要解析与解码的串口输入，则大约每隔 50 ms 就对输出位进行一次更新。

10.5.5 制造

做硬件设计时，我使用了 Fritzing 设计工具，它集成了虚拟面包板、电路图绘制以及易装易用的 PCB 布局设计工具。Fritzing 拥有庞大且活跃的用户群，它提供的库包含大量预载元件，而且最棒的是，它是免费的。一些 Linux 发行版中带有的 Fritzing 版本较老，你可以从官网（http://fritzing.org/home）下载最新版本的 Fritzing 使用。我所用的 Fritzing 的版本号为 0.8.5，它运行在 Kubuntu 14.04 LTS Linux 系统中。对于 Windows 与 Mac OS X 系统，也存在相应版本的 Fritzing 供大家使用。

相比于其他工具，这些年我一直在使用 Fritzing，并且常常惊叹于其用户界面的易用性。老实说，我对它的自动布线器没什么深刻的印象，不过我看到过一些高级的自动布线器对于一些简单的布线表现得也不太令人满意。总之，设计自动布线软件是一个很难的挑战，对它我也不期待会有什么奇迹发生。所以我最终选用手工方式做布局设计（相信你也能看出来）。DRC（设计规则检查）运行得很好，当电路图编辑器表现出一些“怪异”行为时，它也完全可用。对于 Fritzing 软件，我最主要的不满是它的元件库。对于电路图符号的大小，大家的看法似乎都不一致。某些人贡献的库中使用的元件符号可能很小，而 Fritzing 本身提供的元件库中，使用的元件符号真的很大，如图 10-21 所示。最终结论是，如果工具库（包含符号库与布局库）中所有元件不遵守一致的大小约束，并且在不使用网格大小与捕获功能做调整的前提下，很难得到完美的正交线与迹线。

一旦制作好电路图（图 10-21），接下来就可以开始设计 PCB 布局了。第一次做 PCB 布局时没有迹线，只有“鼠巢线”（rat’s nest line），用于指示连接元件的哪些引脚。图 10-28 中，我已经把各个元件放到它们应当处于的位置上，Fritzing 显示出一些“鼠巢线”。在任意一个元件的引脚上点击并按住不放，将以亮黄色高亮显示所有它应该被连接的地方。

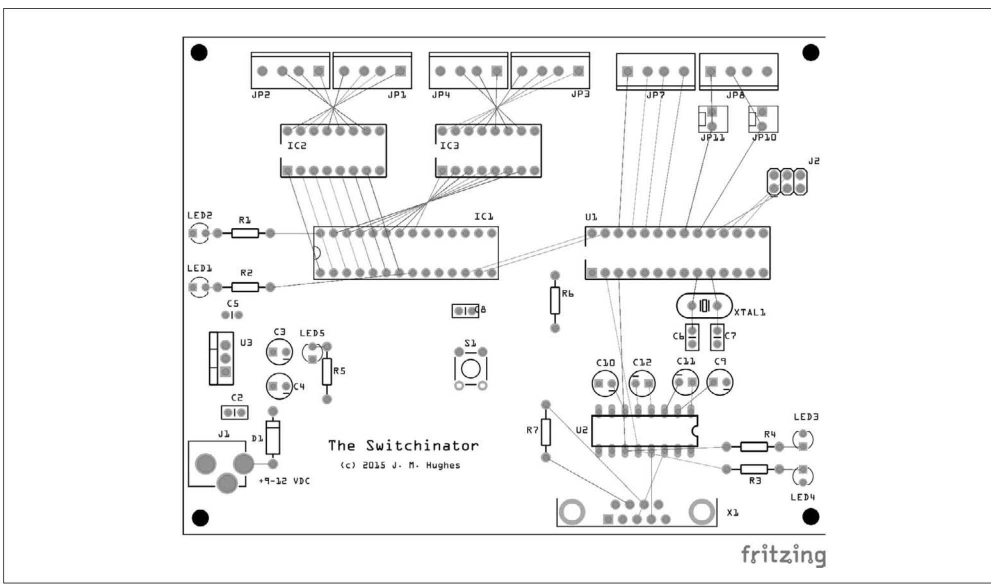


图 10-28：放置元件后、布线之前的 PCB

所有元件按功能分组：电源供给位于 PCB 的左下部分；MCP23017 与两个 ULN2003A 元件位于左上区域；MCU、晶振、ICSP 连接器、模拟输入位于右上区域；串行接口位于右下区域。

你可能已经注意到，我反转了模拟输入接线端子符号以简化接线。但它们的安装仍然是对的，并且由于这种原型 PCB 没有顶层丝印层，所以反转不会造成什么不同。

一些布线相当曲折，主要是因为使用了过孔部件。过孔在 PCB 的正面与底面之间转移迹线，避免布线出现冲突。图 10-29 显示的是最终版本的 PCB，装配制作将基于它进行。

发送 Gerber 文件之前，我使用 Gerbv 工具（该工具在前面学习制作 GreenShield 时介绍过）对设计进行了检查。图 10-30 是当时的屏幕截图。

很幸运，PCB 制作厂（Advanced Circuits）帮忙发现了两处焊盘几乎连接在一起。其实，在工作台上改正这样的问题很容易，真正的困难在于找到这些潜在的问题。有人关注并帮你检查 PCB 的布局设计其实是件好事。

Switchinator PCB 带有过孔，安装起来十分简单。图 10-31 显示的是安装元件之前的 PCB 裸板。

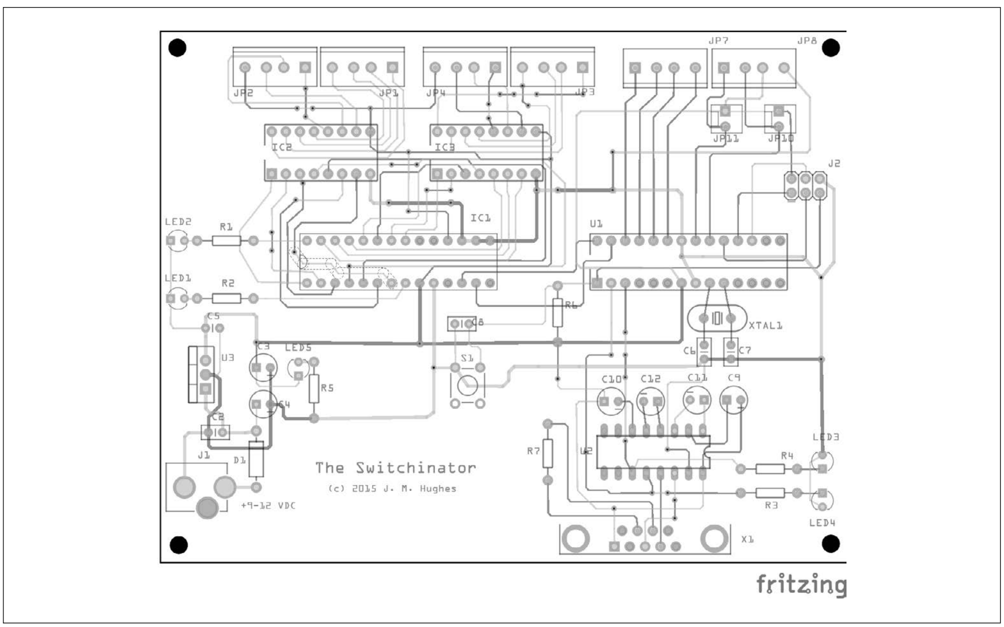


图 10-29：Switchinator PCB 最终布局设计

首先要焊接的元件是 DC 插座与 7805 稳压器，以及相关的电源供给元件，包括 D1、C2、C3、C4 与 C5。请注意，其中并没有 C1（它在早期的修正版本中就没有了）。为了测试电源供给，还需要安装 R5 与 LED 5。安装完这些元件后，接下来就可以通过 DC 电源插座接通 9 V~12 V DC 的电源，并验证稳压器是否可以在 LED5 的阳极正常产生 5 V DC（此时 LED5 应该能够点亮）。

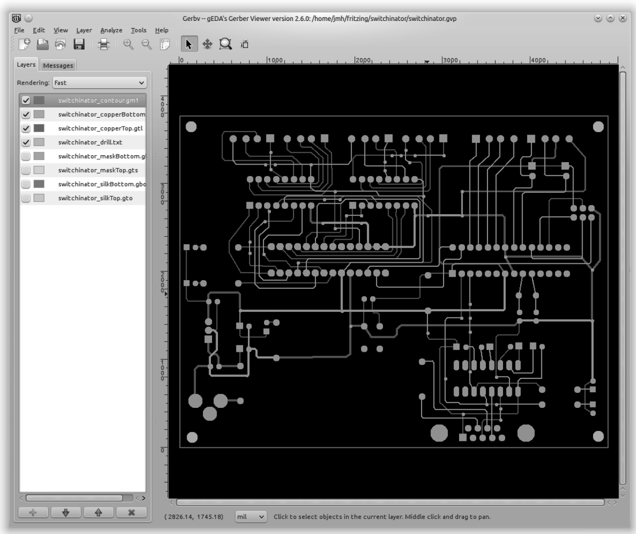


图 10-30：Gerbv PCB 布局图

接下来开始安装 RS-232 连接器（电路图中用 X1 表示）、U2（MAX232 IC）、C9、C10、C11、C12。借助原型面包板，我们可以把来自 MCU 的引脚 2 与 3 的 Rx 与 Tx 信号引出到 PCBU1 的引脚 2 与 3 上。然后，在 AVR MCU 上运行已经准备好的软件，验证集成的 RS-232 是否正常工作。请记得，把 PCB 的地线与 5 V DC 电源连到面包板（当然，操作时要先断开面包板的电源）。

安装好电源供给与 RS-232 接口之后，接着安装 U1（MCU）、IC1、IC2、IC3。同时也可以焊接 LED1、LED2、R1、R2，以及 6 个接线端子。然后安装晶振、C6、C7、C8、R3、R4、R6、LED3、LED4。最后将重置开关、S1、ICSP 连接器安装到 PCB 上。

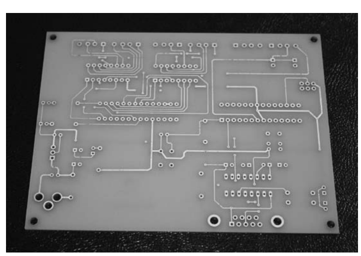


图 10-31：Switchinator PCB 裸板

图 10-32 显示的是安装好的 PCB，其上已经安装所有元件，准备好做最终测试。ULN2003A 驱动器可提供的最大电流由电源供给与 IC 本身的额定值决定。数据手册中指出，ULN2003A 每个通道可以提供约 300 mA 的电流，或者每个 IC 约 2.1 A。相比之下，ATmega328 与 MCP23017 提供的拉电流很小，所以我们主要关心的是 ULN2003A 元件。

如果想用 Switchinator 驱动小型 CNC 工具或者 LED 显示器控制器，使用一个可以提供（至少）5 A 电流的电源即可。

10.5.6　验收检测

很大程度上，对 Switchinator 的最终测试与对原型所做的测试是一样的。它们之间最大的区别是，最终 Switchinator 中有两个 ULN2003A 驱动器与一个基于 MAX232 IC 的集成 RS-232 接口。此外，我们还需要测试电源供给，以及模拟输入。

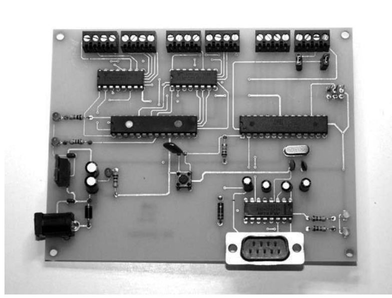


图 10-32：Switchinator 最终成品

10.5.7　后续步骤

其实，MCU 上并非所有离散数字 I/O 引脚都被使用，有 10 个引脚（包括 6 个 PWM 输出引脚）是可用的。由于受到 PCB 空间的限制，我并没有把它们全部引出，但你可以修改 PCB，使用一对六位的引脚插座头。然而，这样就需要你巧妙运用过孔进行布线，避免在 MCU 周围出现凌乱的布线。

SPI 接口可以使用 D11、D12、D13 引脚，这些引脚被连接到 ICSP 连接器。如果你为 SS 线选择一个未使用的引脚，那么可以将 SPI 模块连接到 Switchinator。你可以把模拟引脚用作数字 I/O 引脚，它们对应的引脚编号为 D14~D19。

此外，你可能已经注意到，模拟输入没有保险丝，也没有输入保护。关于模拟输入保护的示例，请参考第 11 章中用作信号发生器的输入电路。

10.6　资源

本章涉及内容十分广泛，但具体内容讲解得并不细致，详细内容需要你自己阅读学习。对于本章简略介绍的一些主题，下面有一些资源对它们做了进一步阐述。

参考图书

市面上有很多图书，涵盖了电子元件的方方面面。下面这些图书是我特别喜欢的，它们与本章所讲的内容息息相关（附录 D 包含这些图书的 ISBN 码，以及更多推荐图书）。

- Jan Axelson，*Making Printed Circuit Boards*

- Paul Horowitz、Winfield Hill，《电子学（第二版）》
- J. M. Hughes，《电子工程师必读：元器件与技术》
- J. M. Hughes，*Real World Instrumentation with Python*
- Simon Monk，*Fritzing for Inventors*
- Matthew Scarpino，《高质量 PCB 设计入门》

电路图绘制与 PCB 布局

Fritzing 与 Eagle 不是唯一可用的电子电路 CAD 工具，但从网上下载其他人创建并推送的电路图或电路板设计时，你常常会遇到它们。Fritzing 免费、开源，除了其自身提供的元件库之外，你还可以从许多其他渠道获得更多元件库。Fritzing 易学易用，非常适合 Arduino 项目。

在免费版 Eagle 中，你可以看到商业版的所有特性。当你想探索专业电子电路 CAD/CAM 工具的世界时，Eagle 为你提供了一条清晰的提升路径。但要记住，使用免费的 Eagle 会有一些限制，并且不适用于以营利为目的的工作。要想在营利目的的工作中使用 Eagle，需要付费购买商业许可。

此外，还有其他一些 CAD 工具是开源的，比如 gEDA 套件、KiCad，它们拥有的功能甚至可以与商业 CAD 工具相媲美。关于这些工具的更多信息，请访问官方网站：

- Eagle（http://www.cadsoftusa.com/download-eagle）
- Fritzing（http://www.fritzing.org）
- gEDA（http://www.geda-project.org）
- KiCad（http://kicad-pcb.org）

PCB 制造商

有许多 PCB 制造商可供选择，大多数制造费用低，周期短。你可以上网搜搜，如果恰巧住在大城市，一定要看看本地有哪些服务可用。我在前面提到了 Advanced Circuits，主要是因为我非常熟悉他们，并且与他们合作时从来没有发生过问题。Fritzing.org 官方也提供 PCB 制造服务（你可以从 Fritzing 工具进行访问）。

- Advanced Circuits（http://www.4pcb.com/bare-bones-pcbs）
- Fritzing Fab（http://fab.fritzing.org/fritzing-fab）

元件购买渠道

我在本书中提到了许多购买渠道，通过这些渠道你可以购买到需要的一切，从单个元件到模块，再到完整的 Arduino 开发板。下面列出的这些公司是你购买元器件时需要优先考虑的（更多建议请参考附录 C）：

- Adafruit（http://www.adafruit.com）
- All Electronics（http://www.allelectronics.com）
- DigiKey Electronics（http://www.digikey.com）
- Mouser Electronics（http://www.mouser.com）
- Newark/Element14（http://www.newark.com）
- SparkFun（http://www.sparkfun.com）

第 11 章

项目：可编程信号发生器

几乎每个电子实验室，无论大小，都需要用到信号源发生器。有时可能是简单的正弦波发生器，有时可能是更复杂的仪器，比如函数发生器（function generator）。本章要讲解的信号发生器可以产生正弦波与方波，最大频率为 40 MHz，如图 11-1 所示。



图 11-1：自制 DDS 信号发生器

当然，你也可以花钱购买信号发生器。它们价格不一，主要取决于频率范围、特性、性能。你可以很容易地为一个设备找到信号与函数发生器套装，价格大约为 6~50 美元，如图 11-2 所示。另一端是商业高端仪器，价格动辄数百、数千美元，甚至更多（比如，一台二手 20 GHz 的多功能任意波形发生器大约为 72 000 美元）。



图 11-2：FG085 函数发生器套件（已组装）

尽管有现成的信号发生器可以产生信号波形，但它们可能不具备你想要的所有特性与性能。毕竟，它们是依据别人的想法制造出来的，而那些人对信号发生器的用途可能有不同看法。亲自动手制造信号发生器，你可以准确把握自己想要的究竟是什么，也可以根据自身需求的变化不断对其进行调整或扩展。

制作信号发生器时，主要考虑的是如何产生信号。换言之，你要决定究竟是用微控制器产生信号，还是将产生波形的工作推给电路中专用的 IC。就 FG085 函数发生器（如图 11-2 所示）而言，它使用 ATmega168 作为仪器的主控制器，使用 CP2101 作为 USB 接口，信号由 ATmega48 与电阻器阵列组成的 DAC 产生。这种方法并无什么不妥，并且允许发生器产生更多波形，而不只是正弦波与方波。事实上，它可以从预载的数据模式（这些数据模式非常有用）产生任意波形。使用微控制器产生信号不利的一面是，必须对设备可产生的最高频率加以限制。对于这个特殊的函数发生器来说，最高限制频率大约是 200 kHz。对于 FG085 元件，或任何一个以这种方式使用微控制器的设备而言，这不是一个缺点（这是一个简单的事实）。

微控制器的运行速度是一定的，它只能跑那么快，其最高时钟频率为它改变输出产生周期信号的速度设置了硬上限。对于许多应用来说，200 kHz 就够了，特别是在嵌入式传感与控制系统中。正如我先前指出的那样，现实世界中事物的变化通常不会那么快，微控制器的时间尺度至少是微秒级。

如果想突破 200 kHz，需要换一种方法。幸运的是，你可以很容易地买到制造信号发生器（同时具备产生正弦波与方波输出的能力，最高频率为 40 MHz）所需的元件，并且它们都是与 Arduino 相兼容的模块。

在 JYE Tech 网站（http://bit.ly/jye-fg085），你可以读到更多有关 FG085 的内容。事先声明，我并不特别认可这个产品，但的确买了一个，它就在我的工作台上，和其他测试设备放在一起。此外，我也有其他信号与函数发生器，有些花哨，有些则不是。在开发与测试新设计的过程中，它们有不同的作用。

平心而论，我应该指出的是，制造本章介绍的信号发生器所需的费用超过 50 美元，而只用 50 美元你就能购买到一个现成的 FG085 函数发生器（如图 11-2 所示）。列出所有需要的元件之后才能算出总的费用，请参考 11.9 节。你需要自己判断这种花费是否合理，充分考虑想对设计及其运行拥有多大控制程度，以及所选用的封装方法。对于我来说，这是值得的，但对你来说可能不是。

请记住，本书讲解的重点是 Arduino 硬件及其相关模块、传感器、元件，书中给出的示例代码都是软件中最关键的部分，它们并不是完整可运行的代码。关于示例与项目的完整软件代码，请前往 GitHub（https://www.github.com/ardnut）进行下载。

11.1　项目目标

对于本章项目，其定义与规划阶段（第 10 章）将合并在一个步骤中进行。从物理结构看，本项目并不复杂，所以我们可以适当精简这些开发步骤，以节省精力，推动项目发展。该项目最复杂的部分是软件，与模块化的微控制器硬件元件协同工作时，软件通常会比较复杂。

本项目的目标是制作一个信号发生器，用作基准测试仪器。虽然它完全可以使用 Arduino 的数字 I/O 产生方波或脉冲，控制某种 DAC，或者使用 PWM 输出产生正弦波的模拟波形，但是其输出频率范围受微控制器的速度限制。不过，还有一种方法可以使用 Arduino 制造信号发生器，其中涉及一款专用的信号发生器 IC——AD9850。

AD9850 是一款“直接数字频率合成”（DDS）芯片，它既能输出正弦波，也能输出方波。你可以对它进行编程，使其产生 0 MHz~40 MHz 的输出。第 9 章已经介绍过 AD9850 模块，它很容易就能购买到，其图片见图 9-57。因为 AD9850 为 Arduino 处理信号生成事务，所以我们可以同时使用空闲的 CPU 周期执行其他功能。除了控制 DDS IC 之外，Arduino 还可以做许多其他的事，比如更新显示、检测门输入、监视操作控制输入等。

这也意味着，信号发生器不需要中断来操纵控制开关。因为 AD9850 一直在运行（除非激活门模式），所以 MCU 有时间轮询按钮开关，并且不会妨碍信号输出。

11.2　定义与规划

本项目的目标是制作一个便携的测试仪器，以便在工作台或类似的环境中使用。它的电源供给来自于墙上的插座，其外壳拥有足够的内部空间，如果需要，将来可以放入电池为其供电。

信号输出：

- 正弦波输出（一直开启），0 MHz ~40 MHz，0 V ~1 V P-P
- 方波输出（一直开启），0 MHz ~40 MHz，0 V ~5 V P-P

函数控制输入：

- 外部门控制输入
- VCO 操作的控制电压输入

用户接口与控制：

- 双线 LCD 显示器
- 频率选择输入（按钮）
- 信号输出电平控制
- 门输入插座
- CV 输入 BNC 连接器
- 电源开 / 关

整个信号发生器将被装入一个带有把手的塑料壳中，如图 11-3 所示。它由墙上的电源插座——又称“墙疣”（wall-wart）——进行供电，带有 9 V~12 V 的 DC 输出。

在前面板上，信号发生器集成了一个双线 LCD 显示器，显示输出频率与状态、各种输入控制、控制电压（CV）与门输入、正弦波（1 V P-P）与方波（5 V P-P）输出。背面板有一个 DC 筒式连接器，用于连接外部电源。



图 11-3：便携式仪器外壳

表 11-1 列出了本项目所需的部分元件，随着项目不断推进，所需元件列表将不断进行调整和完善。这个初始元件列表让我们大致了解项目中要用到什么。在概念与最终成型之间，初始元件列表通常会经历许多变化，有些变化很小，有些则很大。这都是改进设计过程的一部分。

表11-1：初始元件列表

数量	元件	数量	元件
1	Arduino Uno	3	母 BNC 连接器
1	AD9850 DDS 模块	2	香蕉插座（用于门输入）
1	原型开发板	1	壁式电源插座
1	双线 LCD 显示器	1	塑料外壳

11.3　设计

既然有了设计目标与初始元件列表，下面可以着手开始进一步改进设计。首先要明确预期功能，准确定义设备要做什么，以及控制、I/O 需要执行的目标功能。

对信号发生器做什么有了清晰的认识之后，我们把注意力转到外壳上。由于所有元件都要安装到外壳，所以外壳最终决定了我们可以用什么样的显示器、控制输入以及 I/O 连接器。这里的目标是，综合考虑对外壳耐用与便携性的需求以及费用、制作因素，在它们之间尽量达成一种良好的平衡。

因为项目主要使用现成的模块，所以整个项目不涉及电路设计与 PCB 设计的内容。制作过程中，需要将各种模块与信号连接器连接起来，所以要做一些焊接工作，我们将在组装阶段进行。

在信号发生器中，我选用按钮控制输入，而非旋转编码器。这在 DDS 信号发生器的一些设计中比较常见，你可以在很多网站看到，比如 Instructables（http://www.instructables.com）。这样做是因为旋转编码器（虽然很方便用于快速设置某个值）其实只做一件事——测量旋转的数量与方向。而按钮可以做许多不同的事，具体取决于软件在其他控件组成的上下文环境中如何解释它们，以及它们所控制的设备的状态。

11.3.1　功能

本仪器的主要用途是以特定频率（0 MHz~40 MHz）产生信号。AD9850 内部集成了比较器，用于产生同步方波。正弦波是 1 V P-P（peak-to-peak，峰峰值）信号，方波是 5 V P-P。在 Arduino Uno 的控制之下，输出频率在 0 MHz~40 MHz 之间连续变化。通过应用控制电压（CV），可以从外部改变输出频率，并且外部信号（高电平有效或低电平有效）可以调控（或触发）它。图 11-4 显示的是仪器的框图，其中包含表 11-1 列出的所有元件。

图 11-4 的框图中，有 3 个主要元件。Arduino 处理所有控制输入，这些输入可以来自用户，也可以来自门与 CV 输入。LCD 用于显示仪器的当前状态，DDS 模块用于产生正弦波与方波输出。

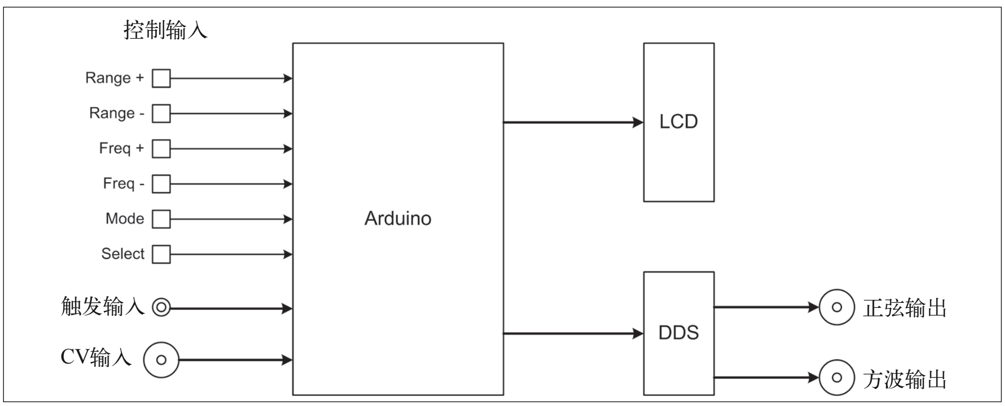


图 11-4：信号发生器框图

图 11-4 中并未给出引脚编号、极性等底层细节，这不是框图的“职责”所在。框图主要以图形形式展示各个部分如何关联以形成某种功能，它也用于检查设计目标是否会超出元件本身的能力。

在图 11-4 中，我做了精心安排，将输出放在右侧，控制输入放在左侧。大圆圈符号表示 BNC 类型的连接器，用于连接屏蔽同轴电缆，小圆圈符号表示香蕉型插座。

双线 16 字宽 LCD 显示器用于显示电流频率及门与 CV 输入状态。一系列控制按钮用于调整频率和外部控制输入的操作。两个电位器用于调整正弦波与方波信号的输出电平。

11.3.2　外壳

本项目中，我选用了 Bud Industries IP-6130 外壳，如图 11-3 所示。这只外壳带有一个手提把手，将其放在工作台上时，手提把手也可以充当支撑架，有时这非常有用。你可以从 Mouser Electronics（http://bit.ly/mouser-ip-6130）下载其数据手册。Bud Industries IP-6130 外壳大约 25 美元，持久耐用，并且看上去很专业，我觉得这个价格还算公道。外形大小如图 11-5 所示。关于 Bud Industries IP-6130 外壳的更多细节，请参考数据手册。

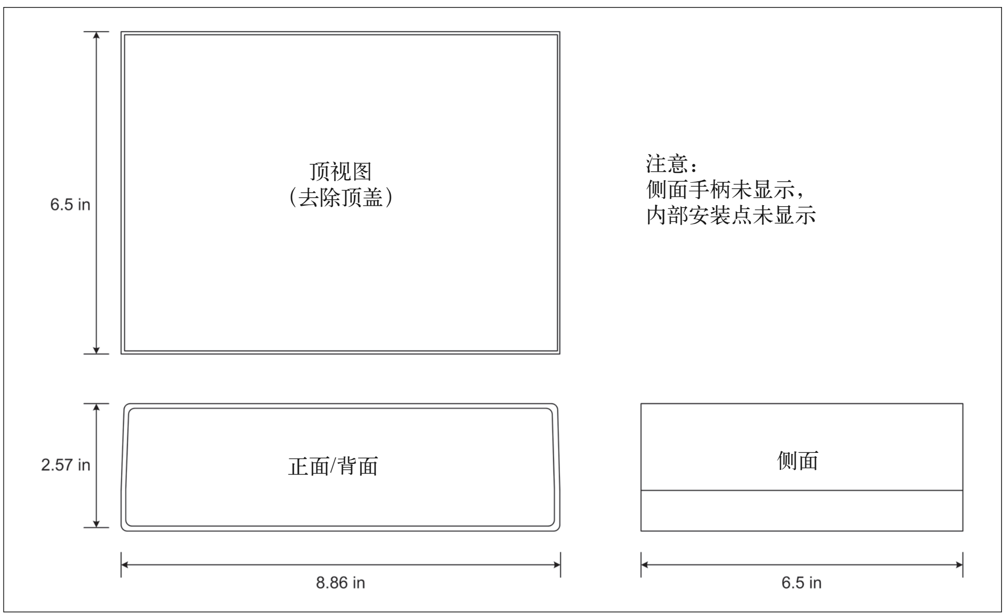


图 11-5：便携仪器外壳大小

需要注意的一点是，外壳的前面板与背面板都不是标准的长方形，顶边比底边窄一些，以便适应外壳边缘的坡度。虽然这不是什么问题，但仍然值得注意，因为只在一个方向上才能正确地安装前面板与背面板。“在面板上打好孔之后才发现原来上下颠倒了”，虽然这算不上什么大灾难，但难免会让人觉得有点蠢（至少在我看来如此）。虽然在面板上剪个洞来指示是否上下颠倒不会是一场灾难，但这样做多少有点尴尬（至少对我来说如此）。那些将外壳顶部与底部结合在一起的螺丝被外壳底部上的粘合橡胶缓冲垫片覆盖着。

另外，还要注意的是，由于前面板与背面板高度不够，所以不能直接将 Arduino 与扩展板堆叠安装在面板里面。IP-6131 外壳更高一些（8.99 cm），但内部会有大量空闲的垂直空间。因此，我决定将 LCD、输入 / 输出连接器、控制按钮安装在前面板，将 Arduino、DDS 发生器安装在外壳底部，将连接外部电源组的插座安装在背面板。这样就有足够的内部空间安装内部电源供给或电池（如果我打算集成其中一个或两个）。图 11-6 描述了前面板的布局情况。

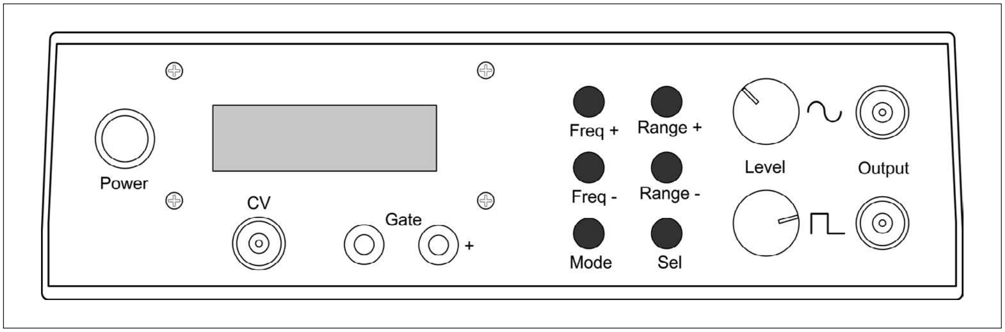


图 11-6：信号发生器前面板布局

另一个需要考虑的是按键开关，它们用于控制输入。那些带有半高方头小按钮阵列的模块在市面上并不容易找到，大多数情况下，它们是为特定产品定制的。由于我们不会进行量产，所以也没必要花时间和金钱设计并制作带有 3×2 开关阵列的模块。最终前面板看上去可能不像图 11-6 那样，但没关系，无论使用什么样的按键开关，它都能一样工作。

11.3.3　电路图

从图 11-4 可以看到，Arduino 上的几乎所有引脚都用到了。事实的确如此，只有 A4 与 A5 引脚未被分配使用。所有数字 I/O 引脚都被用于连接 LCD、DDS、控制开关，并且一些模拟输入引脚也被强行作为数字 I/O 引脚使用。从图 11-7 中可以看到这点。

表 11-2 列出了信号发生器中 Arduino 引脚的分配使用情况。若想进一步扩展 I/O，必须通过 A4 与 A5 引脚连接带有 I2C 接口的扩展板。

表11-2：信号发生器中Arduino引脚使用情况

引脚	功能	引脚	功能	引脚	功能
D0	DDS FQ_UP	D7	LCD E	A0	选择按钮
D1	DDS W_CLK	D8	Range ＋按钮	A1	模式按钮
D2	LCD D4	D9	Range－按钮	A2	门输入
D3	LCD D5	D10	Freq ＋按钮	A3	CV 输入
D4	LCD D6	D11	Freq－按钮	A4	SDA 到 I/O 扩展
D5	LCD D7	D12	DDS RST	A5	SCL 到 I/O 扩展

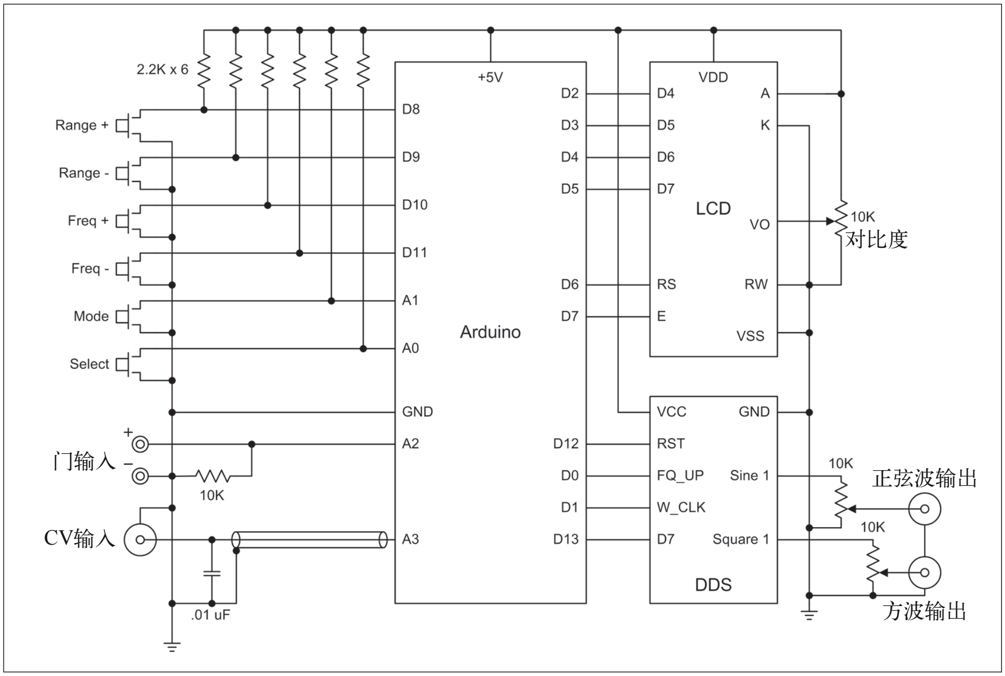


图 11-7：信号发生器电路图

本项目中，我选用的 DDS 模块参见图 9-57。你可以从多个渠道购买，比如 DealeXtreme（http://bit.ly/dx-ad9850），价格大约 8 美元。图 11-8 是 DDS AD9850 模块的引脚图。

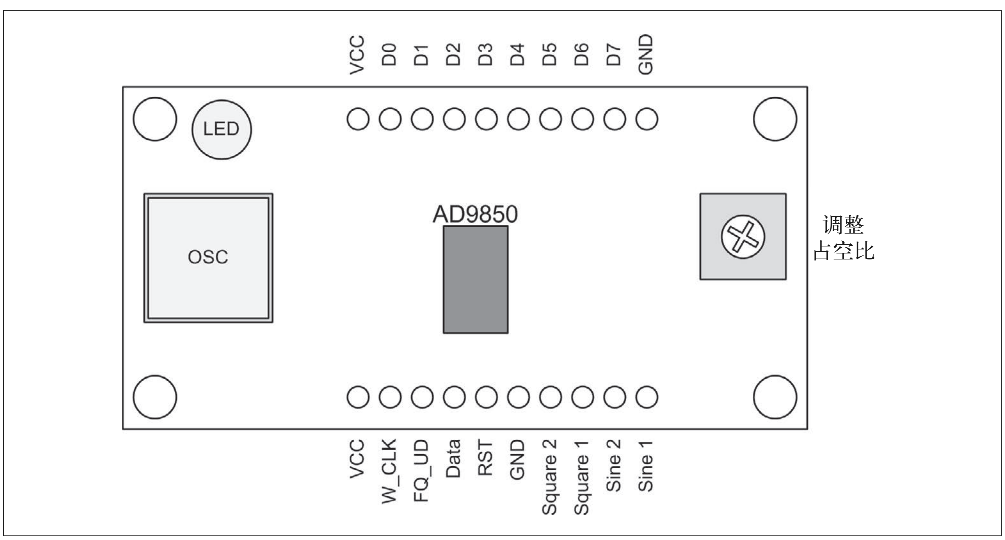


图 11-8：DDS AD9850 模块引脚

AD9850 模块同时支持并行二进制接口与串行字接口。本项目中，Arduino 将通过串行字接口模式与 DDS 模块进行通信。该模式使用 DDS 模块的 D7 引脚接收数据。

LCD 模块有 16 个引脚位，它们将地、电源、数据、控制信号连接到 LCD 控制器 IC，这些控制器 IC 位于 PCB 背面，表面覆盖着黑色环氧树脂。图 11-9 显示了 LCD 模块的引脚定义。

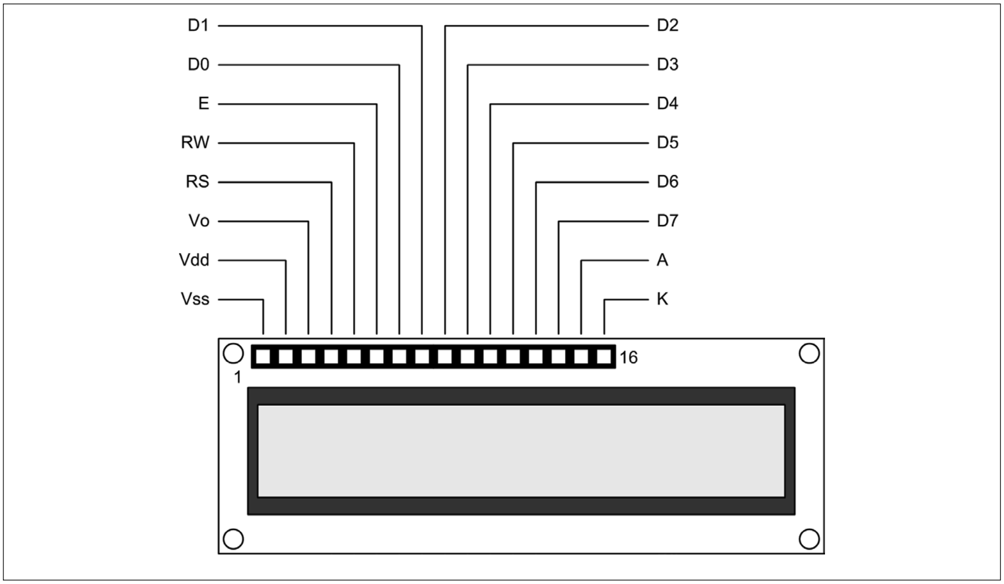


图 11-9：LCD 模块引脚

11.4 搭建原型

本项目使用现成的元件，使用一样的板子与模块搭建原型对开发软件很有用。如果你正在等待购买的元件，那么可以搭建类似的原型以帮助开发软件。对于本项目，我使用了一个测试装置，它由安装在木板上的 Arduino Uno 组成，如图 11-10 所示。

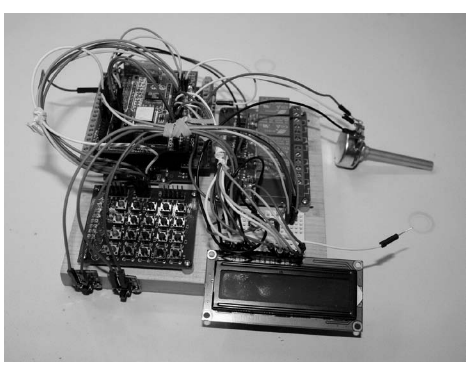


图 11-10：Arduino 原型装置

在原型中，我使用的元器件有 Arduino Uno、带有 5 × 4 微型按键开关阵列的 PCB（包含 8 个 LED）、一对螺丝端子扩展器、一个带有 DDS 模块的原型板、LCD 模块（会在最终产品中使用）。按键将用作信号发生器的控制输入。图 11-10 中的 4 个继电器在本项目中不会使用，所以不会连接它们。关于如何连接 Arduino，请参考图 11-7 中的电路图。

11.4.1 控制输入与模式

前面板上有 6 个按钮，它们对应的功能如表 11-3 所示。

表11-3：信号发生器控制按钮功能

按钮	模式：频率	模式：门	模式：CV
Freq	Inc 频率	导通	启动 CV
Freq –	Dec 频率	断开	禁用 CV
Range	Inc 频率范围	Gate	CV 置 0
Range –	Dec 频率范围	Gate-	CV 复 0

信号发生器有 3 种控制输入模式。从表 11-3 可以看到，3 种模式下总共有 12 种可能的控制输入。此处并未显示选择按钮（Select button），因为它被用作某种 Enter 键。按选择按钮将退出门或 CV 控制输入模式，并将仪器返回到正常操作范围与频率模式下。

波形输出总是开启的，除非被门模式阻止。当仪器不在门模式之下时，可以随时更改频率与范围。按下 Freq + 或 Freq – 按钮并释放，将以 1 Hz 为单位调整频率。当 Freq + 或 Freq – 按钮被按下，并一直处于按下状态时，频率值将以 10 100 或 1000 为单位（取决于当前值）进行变化（增加或减少）。

生成器输出的频率被划分为若干范围，每个范围的跨度为 10 000 Hz。所以，如果范围 1 是 0 Hz~9999 Hz，那么范围 2 将是 10 000 Hz~19 999 Hz，以此类推。使用范围控制不是必须的，这么做的主要原因是，它允许 Freq + 与 Freq – 在一个特定范围内以易控制的增量调整输出频率。否则，用户可能需要不断按 Freq + 与 Freq – 按钮才能调到所需的频率。由于输出频率会被连续更新，因此不必再按选择按钮。

频率范围可以是 1~4000 的任意一个值。如果频率不断增加，超过了某个范围的终值，范围数字将自动增加。如果频率不断减小，越过了当前范围的最小值，则范围值将减 1。范围将自动增加或减小，步长为 10 100 或 1000，就像频率控制输入一样。

将仪器设为合适的控制输入模式后，可以对 CV 与门设置进行调整。模式按钮用于选择命令输入模式，选择按钮用于激活当前设置，以及将仪器重新置为正常操作模式。

当门激活时，生成器不会产生任何输出，除非选择的门条件当前为门输入。Freq + 与 Freq – 按钮用于启用或禁用门。Range+ 与 Range – 按钮用于选择门感知模式，分别为高电平有效（active high）与低电平有效（active low）。

控制电压（CV）输入是一个模拟电压，取值在 0 V~5 V DC，其中 2.5 V 作为标称零点（不用负电压）。在 CV 模式下，Freq + 与 Freq – 按钮用于启用或禁用 CV 输入。选择 CV 模式并按 Range+ 按钮，可以修改零点。使用 Range – 按钮可以将 CV 零点重置为默认的 2.5 V。

一旦启用，CV 输入就一直处于激活状态，直到再次禁用。高于零点的输入电压将引起输出频率增加，而低于零点的输入电压将引起输出频率减小。为了设置零点，要将所需电压应用到 CV 输入，并按 Range+ 按钮。禁用 CV 输入不会改变零点设置。

11.4.2 显示输出

使用小型显示器的最大挑战是，如何高效利用有限的屏幕区域，以简明的形式显示相关信息。信号发生器所用的 LCD 拥有两条位置可寻址线（position-addressable line），每条可显示 16 个字符。图 11-11 描述了运行期间我如何将所有重要信息都显示到 LCD 屏幕。

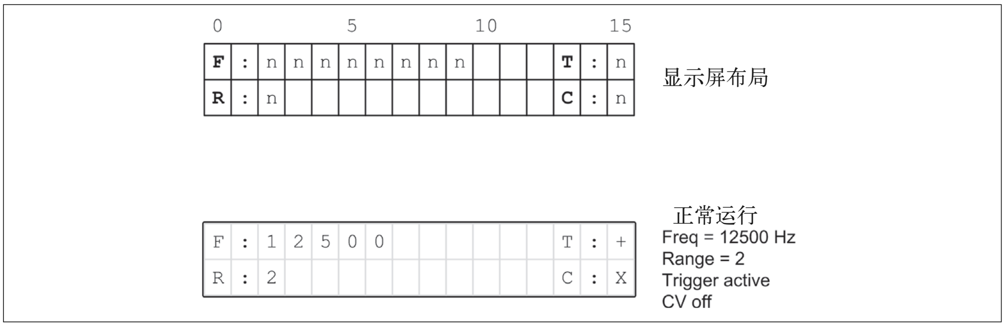


图 11-11：信号发生器操作显示布局

信号发生器处于非频率输出模式时，模式字符后面的冒号将闪烁跳动。所以，如果信号发生器处于 CV 模式，C 字符后的冒号将闪烁跳动；处于门模式时，G 字符后面的冒号将闪烁跳动。

按模式开关（Mode switch）将不断在门（Gate）、CV、控制输入关（正常运行）之间循环切换。只有相关控制模式激活时，才能更改 C 与 G 功能。一旦显示需要的值，选择按钮将把输出设置至所显示的配置，仪器将继续正常运行。

`G`（gate）区域将显示为 `X`、`+` 或 `–`，具体取决于门功能的状态。`+` 符号表示门为高输入（high input）激活，`–` 表示门会响应低输入（low input）。`X` 表示禁用门功能。按下选择按钮将激活门设置。

`C`（CV）区域将显示为 `X`、`0`、`+`、`–`。如果 CV 功能激活，且输入控制电压大于零点（zero point），则显示为 `+`；如果 CV 激活，且控制电压小于零点，则显示为 `–`；如果 CV 激活，且控制电压等于零点，则显示为 `0`。启用 CV 输入时，`0`、`+`、`–` 符号会被实时更新。`X` 表示禁用 CV 输入。按下选择按钮将激活 CV 设置。

从许多方面来说，这可以追溯到众多设备使用小屏的时代。在高分辨率 LCD 与 TFT 显示器出现之前，人们需要使用显示器时，类似的小显示屏很常见。引入字母数字 LED 显示器之前，这些显示器只能显示数字，而解释这些旧显示器绝对是一个技术活。所幸的是，现在我们可以买到既小又便宜的字母数字 LCD 元件，但在数据布局以及某种程度的解释上仍然需要费些心思，用点创意。

11.4.3 DDS模块

AD9850 DDS 模块安装在原型板上，如图 11-12 所示。这不是必须的，你也可以把它安装到外壳底部，并使用导线连接。尽管如此，我还是建议你安装到扩展板。因为对于安装模块来说，扩展板更坚固耐用，并且允许使用螺丝端子。相比于焊接，采用这种方式会让连接更加整洁，并且比其他插入式跳线更可靠。

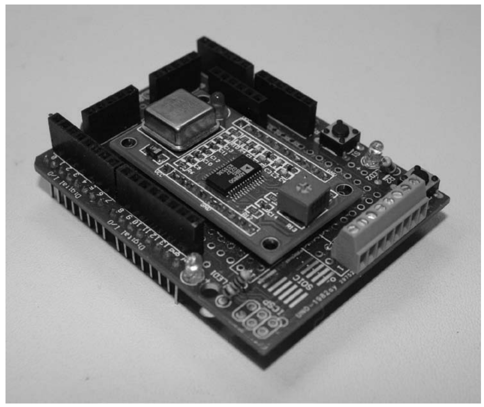


图 11-12：DDS 模块原型板

整个 DDS 扩展板的组成部件如表 11-4 所示。

表11-4：DDS原型板元件列表

数量	元件
1	原型板，Adafruit#51 或类似的
1	8 位 0.1 in（2.54 mm）螺丝端子台
1	6 位引脚插座头
1	AD9850 DDS 模块

DDS 模块的输入连接到 6 位引脚插座头，输出连接到 8 位端子台。电源与地由原型板提供。当前未使用 LEDs，但一个可以连接到电源，其他可以连接到 D7 输入（Arduino D13）。图 11-13 显示如何将 DDS 模块的各种信号线引出到原型板上。

PCB 接线端子与 6 位插座头的连线位于原型板底部。我使用了 28-gauge 的导线，任何在 24-gauge~32-gauge 的导线都可以。连接模块引脚与接线端子时，我没有使用屏蔽电缆，但有时你要考虑使用它，特别是在高频电路中。

我不喜欢将插座头用作 DDS 控制输入，但原型板空间有限，没有太多剩余空间。在最后的组装中，我使用了一些现有的引脚跳线，并剪掉末端，以便插入接收 Arduino 控制信号的螺丝式端子台。最终产品中，我将在这些连接到插座头的导线上覆盖一些透明硅胶，以确保它们安全。

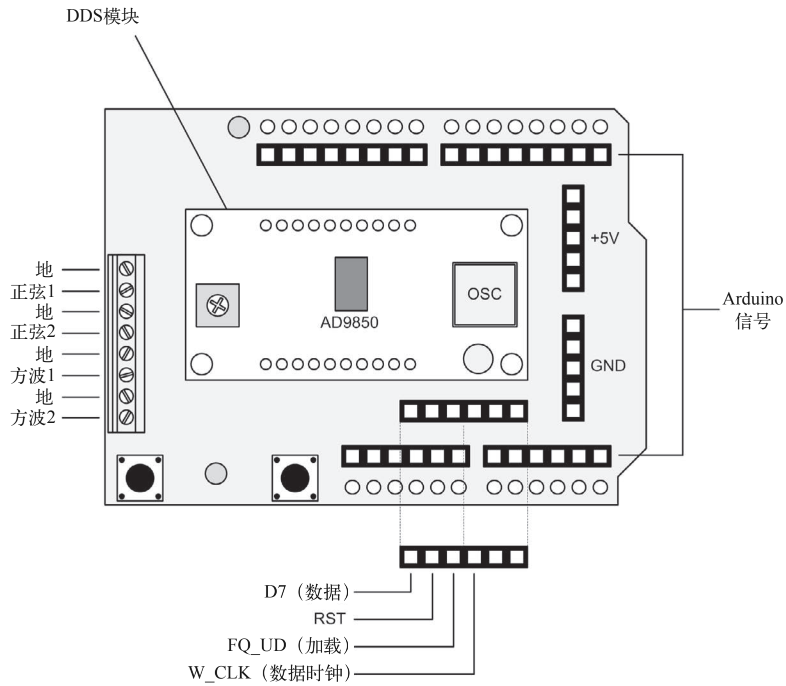


图 11-13：DDS 原型板信号

11.5 软件

编写信号发生器软件时，采用了 Arduino 程序的常见结构，由 `setup()` 与 `loop()` 两个函数组成。其中，`setup()` 函数用于处理 I/O 引脚模式与其他配置事务，`loop()` 函数用于控制输入与管理信号输出。信号发生器软件不同于传统 Arduino 程序的地方在于其组织形式，它由多个源文件（共 6 个）及相应的包含文件（头文件）组成。

复杂度与界面膨胀（Interface Bloat）

从概念上讲，这个项目可能很简单，但令人惊讶的是，软件高度复杂。为什么？这其实是由控制界面引起的。对此，软件开发者与工程师非常熟悉，他们经常使用面向 GUI 的软件，一般有高达 70% 或者更多的代码专用于处理 GUI。这也是 Unix 或 Linux 系统中命令行实用工具功能强大且非常小巧的原因之一，它们没有 GUI 这个包袱要处理，只有命令行界面。

编写代码以实现软件系统与用户的交互时，微控制器应用也遇到“界面膨胀”（Interface Bloat）问题。一般来说，用户比较慢、马虎，并且不擅长记忆与使用紧凑的命令代码，也不擅长解释含糊的响应。所以，MCU 软件的一部分（有时是很大一部分）必须用于创建用户界面，以便有效进行交互。

在一些设计中，这由运行在独立主机系统中的用户界面负责处理。这些主机系统中，内存大小、CPU 时钟频率都不是问题，并通过小巧、高效的面向机器的接口，在主机与 MCU 之间传送命令、参数、状态数据。对本项目来说，这也是一个可能的方案，但不利的一面是，它会削弱 MCU 设备的便携性，因为总是需要通过用户界面连接到主机系统。对于一些应用，这不是主要的考虑因素，比如连接到中央控制计算机的远程传感与控制设备，它们的位置是固定不变的。然而，我们要制作的信号发生器是一个便携的测试仪器，因此需要在它的设计中集成基本的用户界面。

最大的挑战是，找到一种方法，从有限数量的控制输入实现最多设计功能，将有用信息尽可能多地放到有限的显示屏上，同时不要干扰仪器的核心操作，也不要消耗太多可用闪存，以免软件无法加载到 MCU。

事实上，用于控制 DDS 的软件所做的工作只是读取门或 CV 输入，然后将恰当的控制数据写到 DDS IC。最棘手的是如何对控制输入进行映射，以便能够控制这些动作如何以及何时发生。如同其他由微控制器驱动的设备一样，正是软件让硬件拥有了我们所需的功能。如果没有软件，硬件只不过是一堆塑料、导线、电路板、一些硅而已。

11.5.1　源代码组织

信号发生器的代码包含在多个源文件中。在 Arduino IDE 中打开主文件（sig_gen.ino）时，位于同一目录的其他文件也被一起打开，并且以标签的形式展现，如图 11-14 所示。

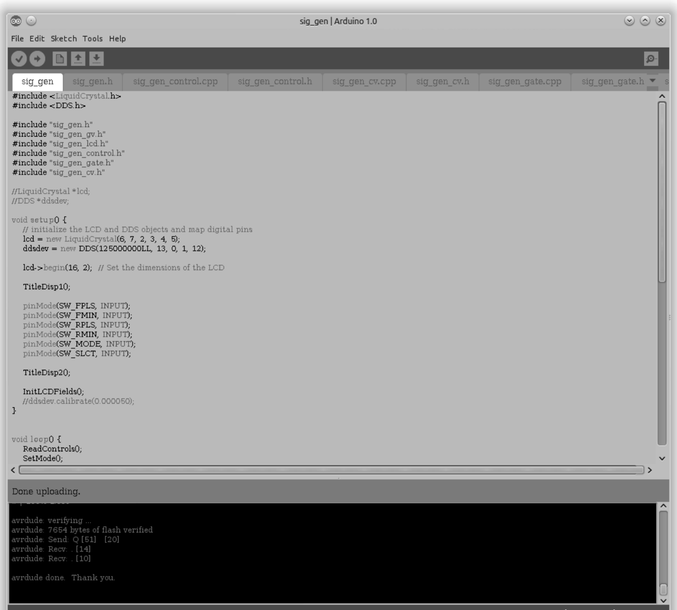


图 11-14：在 Arduino IDE 中打开 sig_gen.ino 程序

信号发生器的代码采用结构化组织形式，比如全局变量全部位于单独的编译模块（sig_gen_gv.cpp）中。LCD、控制输入、控制电压（CV）与门功能也分别位于单独的模块中。表 11-5 列出了源代码模块及其功能。

表11-5：信号发生器源代码模块

模块	功能
sig_gen.ino	主模块，包含 `setup()` 与 `loop()`
sig_gen.h	常量定义（`#define` 语句）
sig_gen_control.cpp	控制按钮输入处理
sig_gen_control.h	包含文件
sig_gen_cv.cpp	CV 输入处理
sig_gen_cv.h	包含文件
sig_gen_gate.cpp	门输入处理
sig_gen_gate.h	包含文件
sig_gen_gv.cpp	全局变量
sig_gen_gv.h	包含文件
sig_gen_lcd.cpp	LCD 功能
sig_gen_lcd.h	包含文件

请注意，在 sig_gen_gv.cpp 文件中，LCD 与 DDS 对象采用 C++ `new` 操作符进行实例化。这是可以的，因为引用变量 `lcd` 与 `ddsdev` 在全局变量模块中定义，并在 sig_gen_gv.h 中导出。为此，主模块必须有包含语句，将 LiquidCrystal.h 与 DDS.h 包含进来，并且同样的包含语句必须出现在全局变量文件中。Arduino IDE 处理作用域与包含语句的方式有些怪异，它要求任何对外部库的引用（包含于标签文件）也必须同时包含在主文件中。此外，还要注意辅助模块包含 `#include "Arduino.h"` 语句，以便模块访问 Arduino 环境。关于 `new` 操作符，5.4 节中的“使用 `new` 操作符实例化类对象”部分介绍过，请参考。

11.5.2 软件描述

信号发生器的代码有点长，此处不会全部列出，只讲解其中关键部分与亮点，并辅以流程图与关键代码片段。我建议你从 GitHub（https://github.com/ardnut）下载全部源代码，并对照下面讲解的内容进行学习。

sig_gen.h 模块由一系列 `#define` 语句组成，定义了程序中使用的常量。像这样，使用 `#define` 语句将某些值定义为常量之后，就不必把这些裸值（naked numbers）直接写入代码。这样一来，修改某个值（比如 LCD 位置或在多个位置使用的时间延迟）就变得很简

单，因为不需要在代码中寻找这些值并逐个修改。并且，逐个修改这些值时，可能发生漏改的情况；而使用 `#define` 语句定义常量后，只需修改该常量的值，之后所有引用该常量的地方的值都会变为修改后的值。当然，也可以使用 C++ 的 `const` 语句，但正如第 5 章讨论的那样，使用 `#define` 语句耗费的内存会更少。

sig_gen_gv.cpp 文件包含全局变量声明，其中包含的代码见示例 11-1。由于 sig_gen_gv.cpp 含有可执行的源代码，所以它可以包含初始化语句。

示例 11-1　全局变量

```
// sig_gen_gv.cpp

#include "Arduino.h"
#include <LiquidCrystal.h>
#include <DDS.h>
#include "sig_gen.h"

// 初始化LCD与DDS对象、数字引脚
LiquidCrystal *lcd = new LiquidCrystal(LCD_RS, LCD_E, LCD_D4, LCD_D5,
                                       LCD_D6, LCD_D7);
DDS *ddsdev = new DDS(DDS_OSC, DDS_DATA, DDS_FQ_UP, DDS_W_CLK, DDS_RESET);

// 替换形式
// LiquidCrystal lcdobj(LCD_RS, LCD_E, LCD_D4, LCD_D5, LCD_D6, LCD_D7);
// DDS ddsobj(DDS_OSC, DDS_DATA, DDS_FQ_UP, DDS_W_CLK, DDS_RESET);
// LiquidCrystal *lcd = &lcdobj;
// DDS *ddsdev = &ddsobj;

// 按钮输入状态值(0或1)
int fpls  = 0;   // freq +
int fmn   = 0;   // freq -
int rpls  = 0;   // range +
int rmn   = 0;   // range -
int mode  = 0;   // mode
int sel   = 0;   // select

// last button states
int last_fpls  = 0;
int last_fmn   = 0;
int last_rpls  = 0;
int last_rmn   = 0;

// 最后按钮状态
int mode_cnt     = 2;      // 模式循环计数器
bool new_mode    = false;  // 若改变模式,则为true
int btncnt       = 0;      // 检测按钮按下
bool btnhold     = false;  // 若按钮按下,则为true
int dds_load_cnt = 0;      // 为DDS更新循环延迟计数器

// 输出控制设置
```

```
unsigned long fval = 1000;
unsigned int rval = 1;

// 显示字符
char gval = 'X';
char gpol = '+';

char cval = 'X';
char cpol = '0';

// 控制循环动作
bool dogate = false;
bool docv = false;
bool gate_output = true;

// CV zero控制
bool set_cvzero = false;
bool reset_cvzero = false;
unsigned long cv_zero = 512;
unsigned long cv_input = 0; // CV输入的最新值

int currmode = MODE_DISP; // 初始化启动控制输入模式

// CV输入
unsigned int cvinval;
unsigned int cvzero;
unsigned int fvalset;
```

请注意，有两种不同方法可以用来创建全局对象。第一种是使用 `new` 语句，第二种是使用指针赋值（pointer assignment）。从功能角度看，它们的最终效果是一样的，但在内存使用方面有一些不同。

包含文件 sig_gen_gv.h 中含有 `export` 声明。导出语句会告诉编译器这些变量将在其他模块中使用，编译的模块包含在 sig_gen_gv.h 文件中声明的变量时，编译器会创建大量占位符。创建最终可执行映像时，链接器会统一处理它们。

信号发生器的输出由一系列全局变量控制，包含电流频率、量程范围、门与 CV 功能的状态。在嵌入式系统中，这是一种常用的方法，特别是没有大量可用 RAM 在栈中存储大量数据的情形。全局变量被用作某种共享内存空间，使用它就无需将大量参数传递给函数。有效使用全局变量的关键是，尽可能应用 write-by-one, read-by-many 规则。在只有一个活动程序（active program）的小系统中，不太可能出现两个进程同时修改同一变量值的情况。但在多线程应用程序中，这是非常现实的问题。

软件中，我们使用 Arduino IDE 惯用的 `setup()` 与 `loop()` 结构。首先了解 `setup()` 函数，图 11-15 显示了 `setup()` 函数的详细流程图。

`setup()` 函数（示例 11-2）很简单。首先初始化 LCD 对象，显示启动信息，为控制按钮初始化数字输入，在 LCD 上显示 Ready，最后将频率、门、范围、CV 字段写到 LCD 上。在信号发生器活动的剩余时间里，这些都会保持不变。

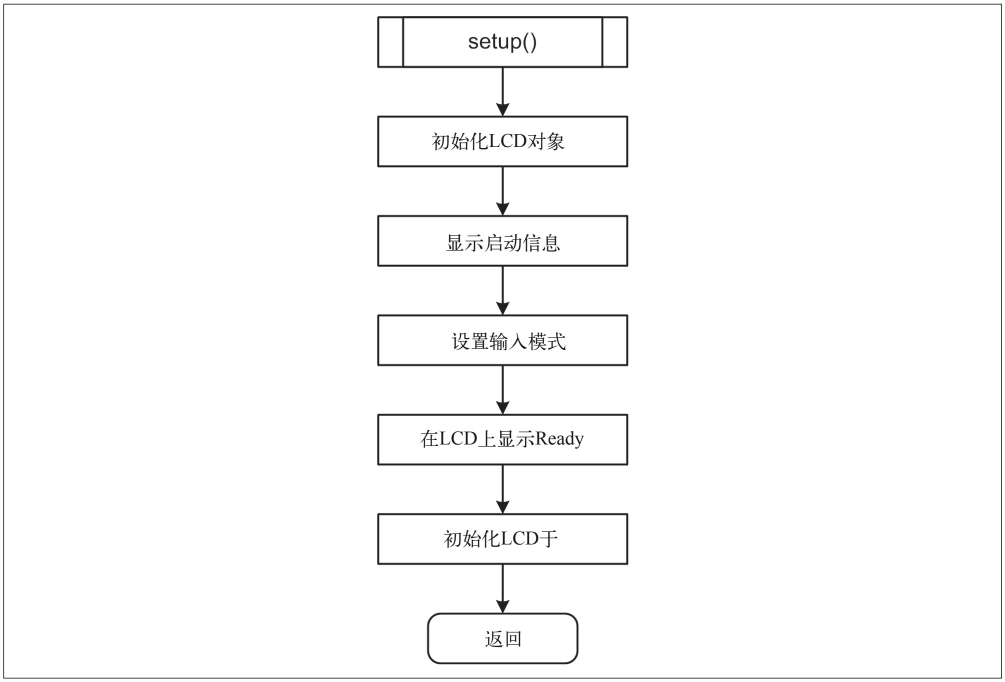


图 11-15：setup() 函数的流程图

TitleDisp1() 与 TitleDisp2() 函数中调用了 delay() 函数，用来延时，防止启动文本一闪而过，导致用户无法读到。

示例 11-2　信号发生器 setup() 函数

```
void setup() {
    lcd->begin(16, 2); // //设置LCD大小

    TitleDisp1();

    pinMode(SW_FPLS, INPUT);
    pinMode(SW_FMIN, INPUT);
    pinMode(SW_RPLS, INPUT);
    pinMode(SW_RMIN, INPUT);
    pinMode(SW_MODE, INPUT);
    pinMode(SW_SLCT, INPUT);

    TitleDisp2();

    InitLCDFields();
}
```

setup() 函数中调用了 InitLCDFields() 函数，可以在 sig_gen_lcd.cpp 文件中找到它。显示器上电完成后，InitLCDFields() 函数将静态字段写至 LCD 屏幕。如果需要，可以在代码的不同位置多次调用该函数。

软件的主循环主要执行如下 4 个步骤：

(1) 检查控制按钮输入；
(2) 检查 CV 输入（若启用）；
(3) 检查门输入（若启用）；
(4) 更新输出频率。

其中，步骤 1 与 2 解析来自主机 PC 的命令字符串，并基于当前模式对控制按钮进行解码。这是软件最复杂的部分。步骤 2 与 3 检查输入，确定要将哪些改变送到输出（若有）。执行简单运算后，步骤 4 将控制数据写到 AD9850。图 11-16 是 `loop()` 函数的高水平流程图（high-level flowchart）。

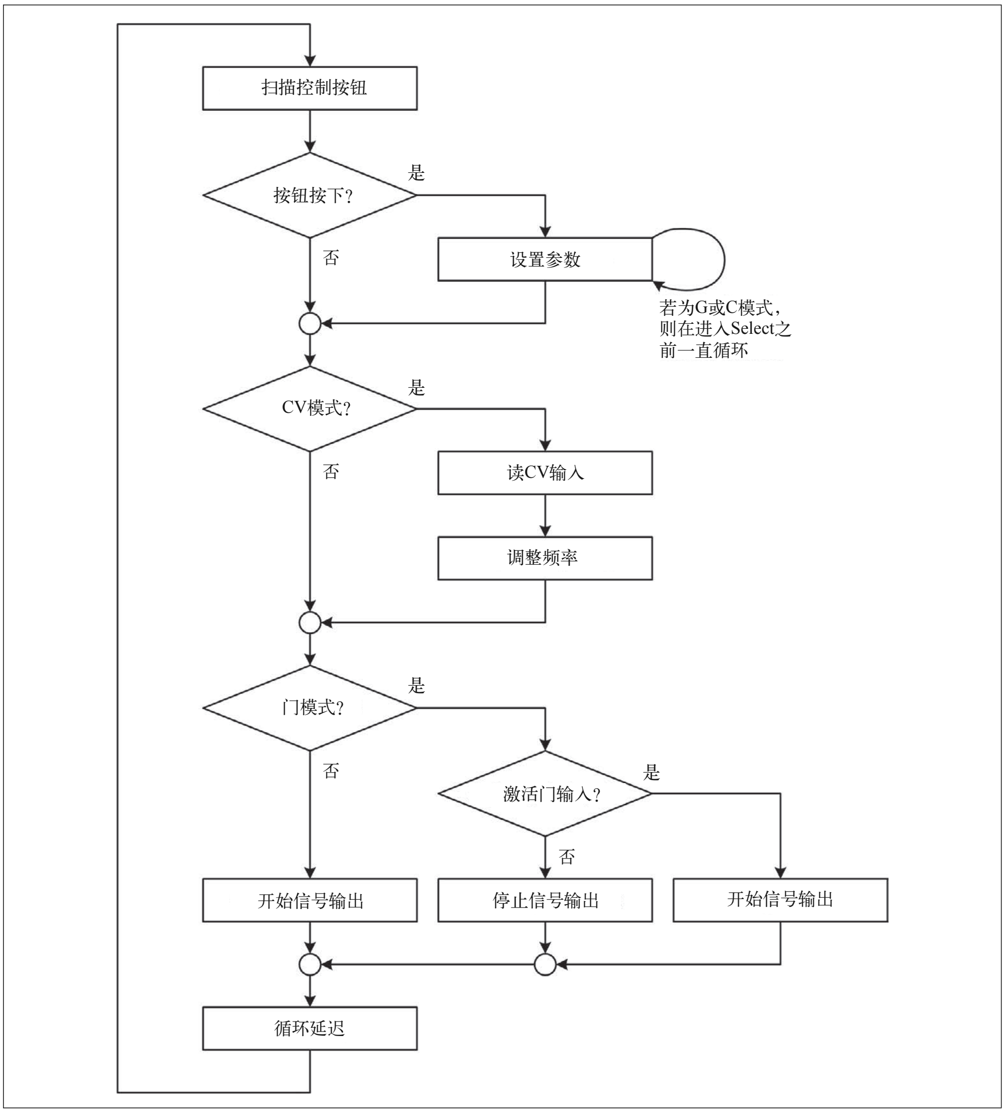


图 11-16：高水平信号发生器流程图（`loop()` 函数）

示例 11-3 列出了 `loop()` 函数的源代码，看上去很简单，因为所有功能的实现都在其他辅助模块中。其中，每个被调用的函数用于设置或读取全局变量。

示例 11-3　信号发生器 loop() 函数

```
void loop() {
    // 读取开关状态
    ReadControls();

    // 处理控制模式
    SetMode();
    switch(currmode) {
        case MODE_DISP:
            getFreq();
            SetLCDVals();
            ShowGate();
            ShowCV();
            break;
        case MODE_GATE:
            SetGate();
            ShowGate();
            break;
        case MODE_CV:
            SetCV();
            ShowCV();
            break;
    }

    // 检查CV输入
    fval = RunCV();

    // 设置DDS输出频率
    dds_load_cnt++;
    if (dds_load_cnt >= DDS_LOAD_GO) {
        dds_load_cnt = 0;

        if (RunGate()) {
            ddsdev->setFrequency(fval);
        }
        else {
            ddsdev->setFrequency(0);
        }
    }

    delay(MAIN_DLY);
}
```

在主循环延迟 `MAIN_DLY` ms 并重新开始之前，所做的最后一件事是控制到 DDS 模块的输出。输出频率（fval）由 CV 输入修改，如果 CV 处于激活状态，则通过 `RunCV()` 函数实现；如果处于门模式状态，则可以通过 `RunGate()` 函数将其设置为 on 或 off。`loop()` 反复循环 `DDS_LOAD_GO` 次之后，才写 DDS 数据，以便为模块留出时间处理来自 Arduino 的控制数据。频率更新间隔（ms）等于 `DDS_LOAD_GO` 与 `MAIN_DLY` 的乘积。如果调整 `MAIN_DLY` 的值，那么可能也要调整 `DDS_LOAD_GO`。

软件中最大的源文件是 sig_gen_control.cpp，其中包含的代码用于检测控制输入开关的活动、去除开关输入抖动、判断用户是否按下按钮，以及设置输入控制模式。

开关往往会产生电气噪声。为了消除它，信号发生器包含了一个简单的 `debounce()` 函数，如示例 11-4 所示。

示例 11-4　Debounce 函数

```
bool debounce(int switchid)
{
    int swval = 0;

    if (swval = digitalRead(switchid)) {
        delay(DBDLY);
        if (swval != digitalRead(switchid)) {
            return false;
        }
        else {
            return true;
        }
    }

    return false;
}
```

`debounce()` 函数背后的思想是，判断开关在两次输入采样之间是否维持相同状态。若是，则开关被认为是处在那种状态下，否则就表示开关有问题。测试时间间隔（`DBDLY`）在 sig_gen.h 中定义，初始值为 10 ms，增加该值可以提升去抖的可靠性，但若设置太大，则会导致控制输入反应迟缓，这可能不是我们希望看到的。

`readControls()` 函数用于扫描 6 个输入按钮开关。如果有开关被按下，那么每次 `loop()` 调用 `readControls()` 函数都会导致计数值增加。如果计数值超过预设值（在 sig_gen.h 中被定义为 `HOLD_CNT`），那么输入就被标记为"被按下"，这通过将全局变量 `btnhold` 设置为 `true` 实现。此外，请记住，`loop()` 函数每隔 `MAIN_DLY` ms 才执行一次。

`getFreq()` 函数用于处理频率输入控制，它会自动增加或减小频率，步长为 10 100 Hz 或 1000 Hz。频率超过范围边界（`RSTEPHz`）时，范围值也会自动调整。

`SetMode()` 函数用于更改仪器当前控制模式，前提是按下模式按钮，并且被 `readControls()` 函数检测到。不断按模式按钮，将在 3 种控制模式（正常、门、CV）之间循环切换。请注意，只有相应模式激活时，才可能修改门与 CV 模式的行为。

信号发生器的完整源代码请从 GitHub 下载（https://github.com/ardnut），代码包含注释，一目了然。

11.5.3　DDS库

信号发生器源代码包含一个简单的 DDS 模块库。为了安装 DDS 库，首先要在 sketchbook/libraries 目录下创建一个名为 DDS 的目录，然后将 DDS.cpp 与 DDS.h 文件放入其中。再次重启 Arduino IDE，将在"项目 -Include Library"菜单中找到 DDS 库（在新版本 IDE

中，不用重启就能立即看到）。你会注意到，DDS 库的子目录文件并不包含 README 与 keywords.txt 文件。因为 DDS 类相当简单，没有必要编写这两个文件。

信号发生器使用的 DDS 库体现了创建自定义库时的一些理念（相关内容见第 5 章）。DDS 类很简单，用于执行必要计算，并为 AD9850 产生控制数据字。DDS.h 文件包含类定义，见示例 11-5。

示例 11-5　DDS 类

```
class DDS {
    private:
        float dds_cal;

        unsigned long dds_clock; // 外部时钟,单位Hz

        uint8_t dds_reset;       // DDS重置引脚
        uint8_t dds_data;        // DDS数据引脚
        uint8_t dds_load;        // DDS数字加载引脚
        uint8_t dds_wclk;        // DDS数据加载时钟引脚

        void pulseHigh(int pin);
        unsigned long freqData(unsigned long freq);
        void sendBit(uint8_t outbit);

    public:
        DDS(unsigned long clockfreq, uint8_t dds_data, uint8_t dds_load,
            uint8_t dds_wclock, uint8_t dds_reset);
        void setFrequency(unsigned long freqval);
        void calibrate(float calval);
};
```

实例化 `DDS` 对象之后，如果想提高信号发生器的精确度，可以向其传入一个校准系数。关于如何计算该值，请参考 AD9850 的数据手册。本类将简单地采用默认值 `0`。

除了对象的构造函数与 `calibrate()` 函数之外，类外部代码可以调用的唯一函数是 `setFrequency()` 函数。频率单位为 Hz，经过计算得到的控制字被推送到 DDS IC，每次 1 位。

11.5.4　测试

假如原型的所有连线都正确，那么应该能够编译并上传软件到 Arduino Uno 中。启动后，显示屏上依次显示 DDS Signal Gen、Initializing、Ready，然后是数据显示字段。可以调整延时时间，让启动文本停留时间更长；或者干脆删掉，这完全取决于你自己。

> 如图 11-7 所示连接信号发生器时，不能使用串行库。该库会占用 Rx 与 Tx 引脚（依次对应于 D0 与 D1），并把它们上拉。这会搞乱 9850 DDS，使之不产生任何输出。串行库适合用于调试控制输入，就像我做的那样。但连接了 DDS 时，千万不要将其实例化。可以重新分配模拟与数字引脚，将 A4 与 A5 用作控制开关输入，将 D0 与 D1 用作串行 I/O，但我认为在本项目中没有必要这样做。如果需要将新软件上传到 Arduino，使用 USB 接口就够了，甚至在有 DDS 模块接入的情形下亦可。

为原型上电后，代码默认产生 1000 Hz 输出。这可以是 AD9850 允许范围内的任何一个值，随你选择。但此处我选了 1000，因为我可以轻松使用任何一种示波器看到它。

> 为了查看 DDS 输出，你需要准备一台示波器，频率越快越好。Seeed 有一些便宜的单通道或双通道 Nano 数字示波器，外形类似于 MP3 播放器，一般频率不会超过 1 MHz。如果想查看 DDS IC 频率范围的高频段，需要一台带宽至少为 100 MHz 的示波器。我有一台小的 Seeed DSO Nano，方便快速查看低频电路。如果要查看高于 1 MHz 的电路，需要使用更强大的示波器。

首先要检查的是正弦波输出。图 11-17 显示的是频率为 1000 Hz 的正弦波，振幅约为 1 V P-P。原型中没有输出级别控制，但最终产品中会有，所以如果没有看到正弦波，试着拔掉那个部件，并重新检查相关连接。

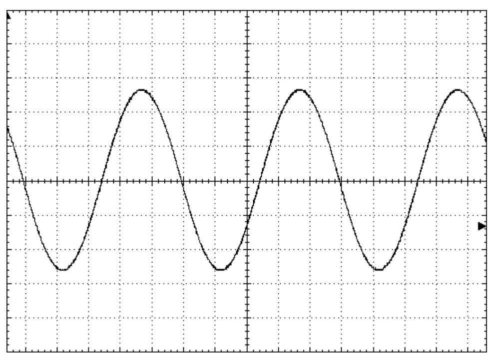


图 11-17：默认输出 1000 Hz 正弦波

方波输出的频率与正弦波相同，振幅约为 5 V P-P（其实只有 V+），输出如图 11-18 所示。

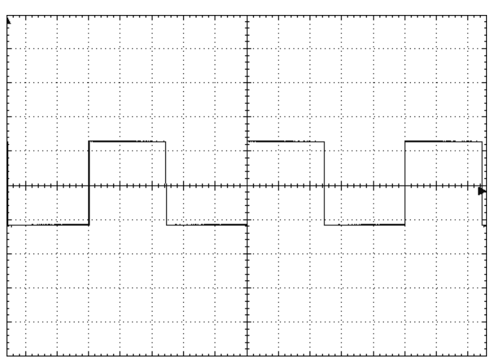


图 11-18：默认输出 1000 Hz 方波

如果同时存在两种输出，那么可以继续测试控制输入。通过按 Freq+ 与 Freq – 按钮，可以测试频率与范围输入。每按一次按钮，显示屏上的频率就会以 1 Hz 单位进行改变。现在按住 Freq+ 按钮不放，频率先以 1 增加，然后以 10、100，最后以 1000 增加，大约增加到 15 000 停止，检查输出。请注意，频率值超过 9999 时，范围值将自动增加。按住 Freq – 按钮将重复相反过程。随着频率的减小，范围值也随之减小。

按 Range+ 按钮，每按一次，频率将以 10 000 为步长增加。将范围值改为 4 并观察输出，应该显示 40 kHz 左右。按 Range – 按钮，观察显示，频率将逐渐减小，在范围值内每次以 10 000 为单位减小。

按下模式按钮不放，直到符号 G 之后的冒号开始闪烁跳动，释放模式按钮。然后按 Freq + 与 Freq – 按钮，可以启用或禁用门输入。按 Range + 与 Range – 按钮改变输入感知状态，它应该在 + 与 – 之间循环变动。按选择按钮，返回正常操作模式。

当门处于激活且没有门输入时，输出会停止。用一根跳线将门输入（在 Uno 上是 A2）连接到地时，选择 –（低电平有效）输入模式，可以观察到输出被激活。在 +（高电平有效）输入模式时，使用跳线连接到正电压源，输出将被启用；而当 A2 输入接地时，输出将停止。通过进入门控制模式并使用 Freq – 按钮禁用门操作，可以禁用门输入。

为了测试 CV 输入（Arduino 的 A3 引脚），需要准备一个可变电源，或者一个 5 V 电源与一个 10 K 电位器，以及一个 DMM，如图 11-19 所示。如果示波器不支持频率显示功能(大部分现代 DSO 仪器都支持)，频率计数器会非常有用。

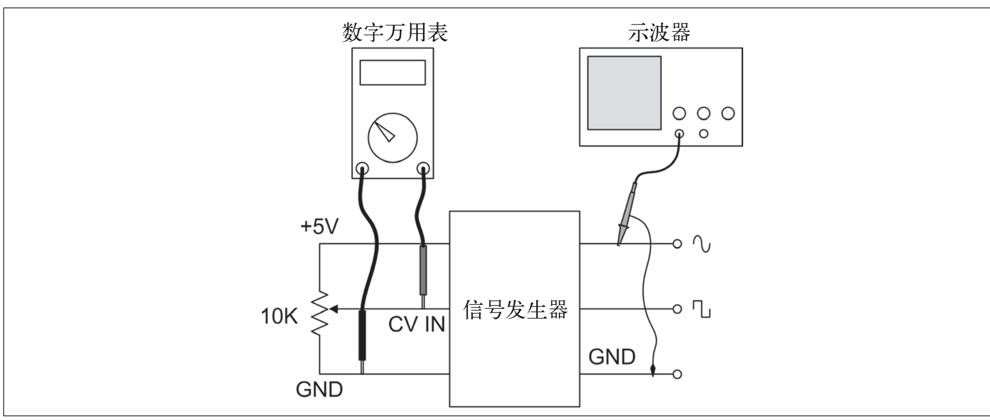


图 11-19：控制电压输入测试装置

通过模式按钮启用 CV 输入，但不设置零点（保持为默认的中间值，约为 2.5 V）。按选择按钮返回正常操作模式。观察频率输出时，施加一个 0 V~5 V 的电压。CV 输入低于 2.5 V 时，频率降低；大于 2.5 V 时，频率将增加。通过观察 + 与 – 极性指示，可以知道 CV 输入的极性。在零点手工设置 CV 几乎是不可能的，但输入超过零值时，可以看到显示器短暂显示为 `0`。

现在调整 DMM 上的 CV 输入电压，约为 2 V。将仪器置于 CV 模式之下，启用 CV 输入，按 Range+ 按钮设置新的零值。此时，极性符号应该是 `0`，只要输入电压不改变，它将一直

保持为 0。改变输入电压可以观察到，随着 CV 输入在新零点值上下变动，输出频率也随之上下改变。测试结束后，使用模式与 Freq – 按钮禁用 CV 输入功能。

请不要向 CV 输入施加大于 5 V 的或为负的电压，因为这个原型电路没有任何输入保护。一旦超过 ADC（位于 AVR MCU）允许的输入范围，ADC 就会被烧毁。在最终产品中，我们将为它设置一个简单的输入保护电路。

基本功能测试到此为止。如果一切工作正常，接下来应该测试信号发生器，看看它如何响应控制输入。这也是设置方波占空比的好时机。

DDS 模块的 PCB 上有一个小的 PCB 电位器，与银色振荡器模块相对。将频率设置为 10 KHz 左右，并通过示波器调整方波，以便让 on 与 off 区段所占时间相等。在最终产品中，你可以拆除 PCB 上的电位器，使用一个装在面板上的控件来代替它。该控件安装在信号发生器前面或背面。

11.6 最终组装

如果打算为 DDS 模块选用原型板，那么建议你选择带有螺丝端子扩展器的板子，比如 8.4.21 节列出的那些。使用它们不仅可以保证连接的可靠性，还可以将 DDS 板置于 Arduino 之上，为访问 ICSP 连接器留出空间。此外，也可以使用螺丝终端板，像 11.8 节中展示的那样。

11.6.1 上拉电阻阵列

安装在原型穿孔板上的 6 个 2.2 K 电阻被用作 6 个按钮开关的上拉电阻。虽然 AVR MCU 拥有一定程度的上拉能力，但电阻阵列可以确保有正电压供开关使用。同时也意味着，开关输入是低电平有效的（0 V 输入 =on）。

从电气结构看，上拉模块非常简单，如图 11-20 所示的电路图。6 个电阻分别连接到各个控制输入开关的信号线上，6 个电阻的另一端统一连接到 +5 V DC。

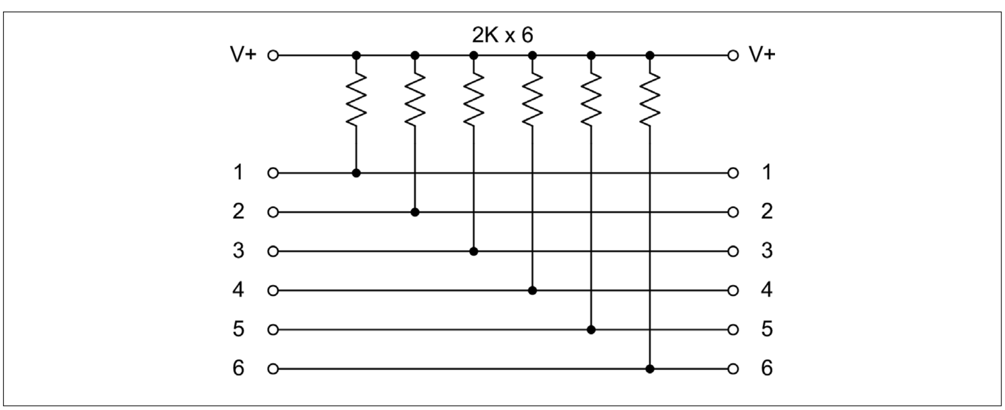


图 11-20：控制输入开关上拉电阻阵列电路图

最终，上拉电阻阵列板如图 11-21 所示。使用 0.1 in PCB 接线端子可以让连接更容易，同时保证连接的可靠性。从图 11-21 也可以看到输入保护模块以及与 DDS 原型板安装在一起的 Arudino，此时只连接了 +5 V 与地。

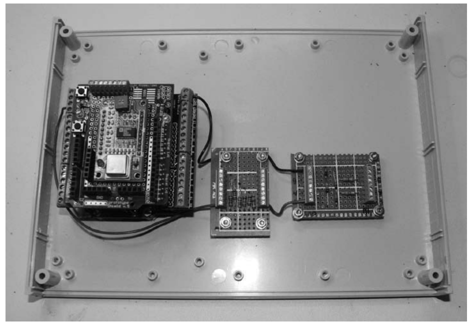


图 11-21：上拉电阻阵列与输入保护模块

11.6.2 输入保护

一个小的穿孔板模块用于构建简单的输入保护电路，如图 11-22 所示。该电路阻止任何高于 +5 V DC 或低于 0 V（比如负电压）的电压进入门或 CV 输入。

470 Ω 的限流电阻串联在输入端，它可能导致 AVR ADC 输入端的电压略微下降。但由于门充当二进制输入，CV 是一个相对输入，所以无妨。如果担心二极管在过压或欠压情形下流经的电流太大，甚至可以增加电阻值。最终保护模块如图 11-21 所示。此处还使用了 PCB 接线端子。

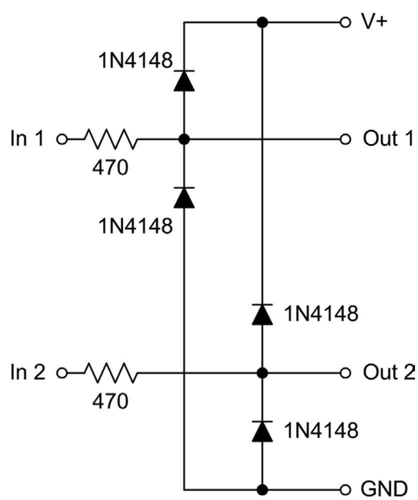


图 11-22：外部控制输入保护电路图

11.6.3 机箱外壳

至此，我们已经编写好了软件，并准备好了各种组成元件，接下来要做的是就是把它们装入机箱外壳。在这一过程中，最复杂的部分是在前面板钻孔，并且保证这些安装孔的位置大小正确。为此，我决定从前面板的背面开始钻孔，这有助于保持面板干净、整洁。当面板被工具意外划伤时，因为在背面，所以没人会看到。接着，将 Arduino 与 DDS 模块安装到外壳底部，然后将 USB、DC 电源连接器安装到背面板。最后，适当连接各控件、连接器、模块。

首先，在前面板与背面板钻孔。在前面板上要凿出一个矩形区域，用于安装 LCD 显示器；并钻多个孔，用于安装开关与连接器。背面板要安装板式 DC 电源连接器，以及用于调节 LCD 对比度的电位器。Arduino 的 USB 未被引出到背面板。移除外壳盖之后，才能访问 Arduino 的 USB 连接器。图 11-23 展示的是使用小型钻床在前面板钻孔的场景。当然，也可以使用大型钻床钻孔。此外，还可以使用手钻，但要先在钻孔位置做个标记，防止钻错。钻孔时不要着急，先从小的启动孔开始。我先使用机工尺与游标卡尺标好孔洞的位置、大小（见图 11-24），然后使用电动雕刻机与小钻头打定位孔。



图 11-23：在前面板钻孔

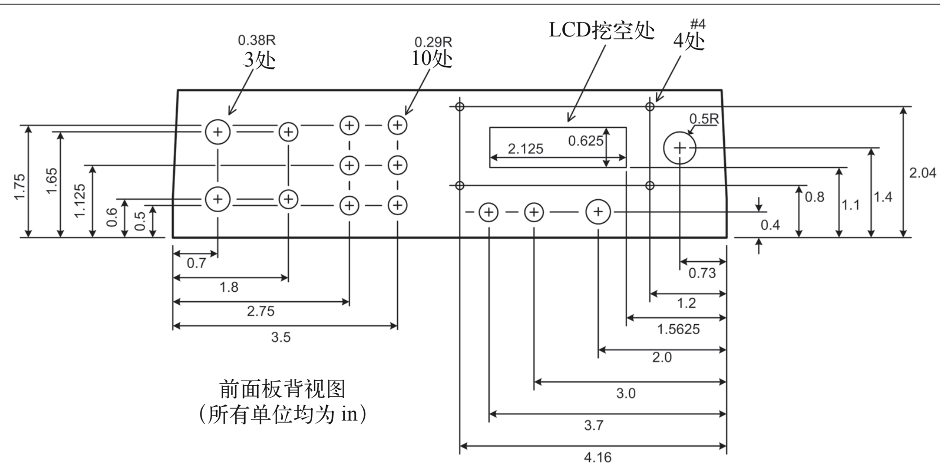


图 11-24：信号发生器前面板安装孔布局图

图 11-24 是信号发生器前面板的钻孔分布图。我指定 #4 孔用于安装 LCD 模块，你也可以使用更小直径的安装孔，但要保证位置精确无误。背面板只有一个 DC 筒式电源连接器与一个用于调节 LCD 对比度的电位器，只要将其安装到你认为合适的位置即可。

制作本项目时，我耗费了一些精力来绘制详细图纸。虽然这不是必需的，但这样做具有长远价值。设计与加工图纸记录了要做什么，使用它们可以避免重复设计工作。除非有重大修改，否则可以一直使用这些图纸制造更多的信号发生器。

Arduino Uno 与 DDS 原型板安装在外壳底部，同时安装在此的还有上拉电阻阵列与输入保护模块。具体安装位置不是特别重要，但 Arduino 应该安装在靠近前面板的位置上，以便保持连线整洁、有序。图 11-21 显示了内部安装元件，以及准备连接到前面板的元件。

安装好内部元件以及在前面板钻好孔洞之后，接下来要做的是将各种控制器件与 LCD 显示器安装到前面板。请不要忘记，在各个控件与连接器上打上标签。

> 如果你打算使用贴标机，那么可以在安装控件之前或之后印制各种标签。然而，如果打算使用转印贴纸干印，或者采用某种方式（比如丝网印刷）将印字喷涂到面板上，那么应该在安装元件之前做这项工作。另外，当你使用贴标机时，请确保已经为设备准备了合适的印字胶带。比如，某些机型没有可用的黑底白字印字胶带。使用贴标机之前，请确保你能获得所需的印字胶带，或者有其他替代方案。我使用激光打印机与背胶标签印制文字（黑底白字），然后剪成需要的大小并贴到前面板上。

一旦将所有元件放入适当位置，并在各种元件之间做好连线，你会惊讶于用元件填充外壳的速度。图 11-25 显示的是合上顶盖之前信号发生器内部的样子。请注意，背面板上安装着用于调节 LCD 对比度的电位器，我并没有费力地为它安装旋钮。

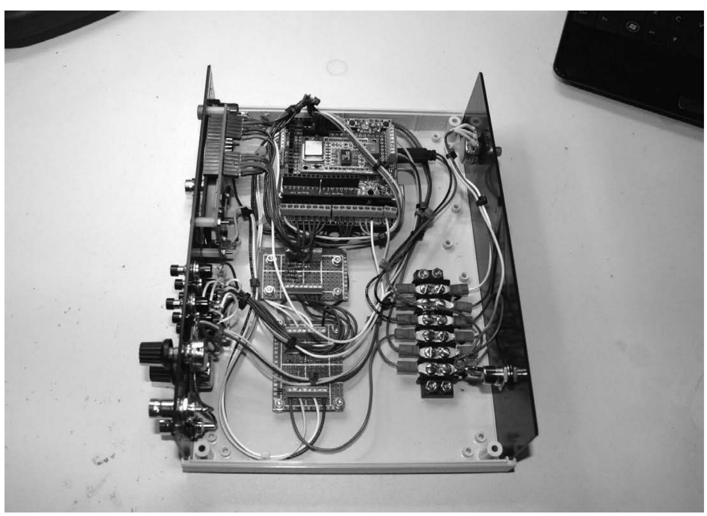


图 11-25：组装好的信号发生器

现在，我们还没有为信号发生器装顶盖。完成最终测试之前，请一直开着。人无完人，制作信号发生器过程中，我们可能会犯一些小错误。如果顶盖开着，那么出现错误时，我们就能很方便地进行排查，拧紧松弛的螺丝或重新连接线路。

安装 LCD 模块时，用到了 4 根绝缘管，在螺母下配有开环锁紧垫圈。我从前面把标准的堆叠插口头焊接到 LCD PCB，然后修剪后面导线约 1/4 in（6.5 mm），以便将插口头压到引脚上。那些焊接到引脚的连线受到热收缩管保护，为 LCD 模块构成相当不错的自制连接器。在图 11-25 中，你可以看到这些 LCD 连接器。

我使用束线带让连线保持整洁，使用 PCB 接线端子让连接变得轻松、容易。稍微麻烦的一点是找到一种方法，将 DC 电源输入与背面板的 LCD 对比度控制器连接起来。为此，我安装了一个 6 位的接线端子，它带有 #6 螺丝与压接片式端子。

11.6.4　DC电源

信号发生器的背面板上安装有一个筒型连接器，用于连接 DC 电源输入。除了 DC 电源插座之外，我还购买了连接插头。这样就可以选用带有合适电压输出的变压器（DC 适配器）、转换插头，以便正确连接电源连接器。来自背面连接器的 DC 电压先到接线端子，再到前面板电源开关，最后到 Uno 开发板上的 DC 连接器。由于 DC 电源不会连接到与地相接的外通路中，所以我并没有为它安装保险丝，但安装一下也并不是什么难事。

针对不带 USB 连接的 Arduino，官方推荐使用 9 V~12 V DC 适配器。使用 5 V DC 适配器时，Arduino PCB 上调压器的电压会有大幅下降，5 V 端子只能读到 3.5 V 左右。我修改了一个现有的 9 V DC 适配器，它来自于我的一个大的剩余“墙疣”盒（随着时间的推移，这些东西的积攒速度真是惊人）。关于 DC 适配器的更多内容，请前往 Arduino Playground（http://bit.ly/apg-what-adapter）阅读相关资料。

11.7　最终测试与结束

既然一切都已安装在外壳中，各个标签也已贴好，软件亦加载完毕，接下来就该对信号发生器做最终测试了。整个测试过程其实只是简单的回归测试，测试步骤与测试原型时所用的那些步骤一样。测试的主要意图是，验证原型与最终产品之间没有发生什么变化，也没有出现任何错误。

若要做进一步测试，则需要使用一些专业设备，比如失真度分析仪或频谱分析仪，而大部分人是没有这些设备的。如果你碰巧认识拥有或者有权使用这种测试设备的人，那么一定要好好把握机会。有时，了解各种频率下谐波失真程度（恒定的还是会随着输出频率改变）以及正弦波输出的纯净程度也许很有用，这主要取决于你打算如何使用自己的信号发生器。其他需要检查的有方波输出的上升与下降的时间、输出阻抗、CV 输入的频率电压（hertz-to-voltage）关系、长时间稳定性（比如长时间运行时频率是否会发生漂移）、CV 与门控制输入响应时间。

完成最终测试之后，进入最后一步，即为信号发生器安装顶盖。不要忘了在外壳底部安装

粘合橡胶垫，注意不要让它们盖住箱盖螺丝。图 11-26 展示的是最终组装完成的信号发生器，它就放在我的工作台上，同时还有一个 USB 数字示波器与一台笔记本电脑。

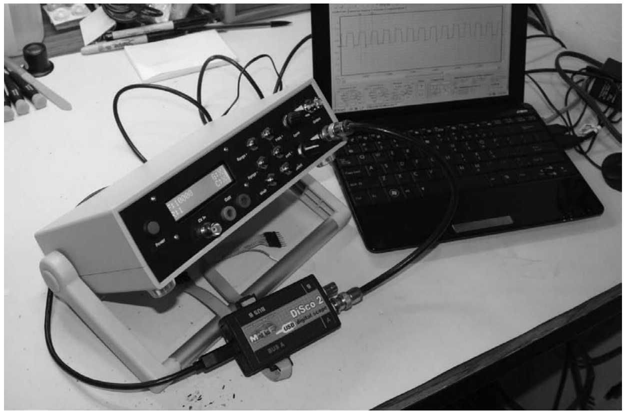


图 11-26：组装好的信号发生器

我必须提及，从图 11-26 可以看到，信号发生器的把手有点怪。我让外壳的顶部与底部方向保持正确，但意外颠倒了前后位置。对此，我不打算狡辩，说什么“这是有意为之”，因为这很明显是我的错误，但实际上它仍然工作得很好。正常情况下，把手应该从设备的前方伸出，以便更好地支撑设备。在后方把手的支撑下，发生器仍能放在一个倾斜的位置上，并且这样也不会占用我的工作台太多空间。我很幸运，这个错误并没有造成什么灾难。但并非所有错误都如此，绝大多数错误都会导致不可预知的后果。

11.8 削减开支

如前所言，信号发生器不是一个便宜的项目，其中的部分原因在于选用了价格较高的外壳、连接器与控件，以及其他一些附件，比如螺丝端子扩展器、上拉电阻阵列、输入保护模块。如果你想削减开支，下面提供几种有用的方法，以帮助你稍微降低一点成本。虽然钱花得少了，但仍然能买到一些能用的东西。

我发现有一些 Arduino 克隆板很便宜，只有 15 美元。本项目中，DDS 模块必不可少，所以你需要花 10 美元左右购买。带有螺丝端子的原型板大约 16 美元，但你可以选购一个更简单的原型板，只带有 DDS 模块安装位，大约为 10 美元。为了进一步削减开支，你完全可以不使用外壳，只要在 Arduino 上堆叠板子即可。市面上有一些 LCD 板本身就带有按键（比如 8.4.19 节中提到的那个），每个大约 12 美元。一般而言，这些板子只有 4~5 个按键可以实现编程功能，所以你可能需要重新设计控制界面，以适应更少的控制输入。

如果不需要或者不想设计成单独的设备，那么也可以省去 LCD 与按键控制器，整个发生器只由 Arduino 与 DDS 模块两部分组成，且在主机控制下运行。这是一种不同的设备，本章不做讲解。关于如何实现远程控制界面，请参考第 10 章中 GreenShield 与 Switchinator 项目的软件。

也可以通过堆叠开发板的方式创建信号发生器，这些板子包括一个 Arduino 板、一个原型板（用于安装 DDS 模块）、一个 LCD 扩展板，如图 11-27 所示。

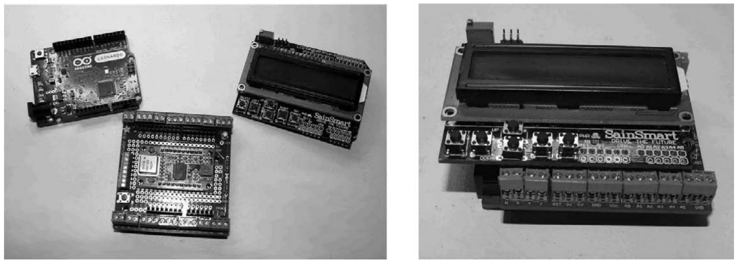


图 11-27：廉价版信号发生器

其中，Arduino 板是指 Arduino Leonardo 开发板，原型板来自 Adafruit，它是螺丝端子原型板，其上安装有 DDS 模块；LCD 扩展板来自 SainSmart，板子本身带有按键。一个 0.1 in（2.54 mm）的 PCB 端子台被安装到原型板上，用于从 DDS 模块获取信号。除此之外没有连接器，没有输出电平控制电位器，也没有外壳。请记住，缺少保护壳的情况下，如果恰巧有零散的导线或螺丝起子碰到上了电的扩展板堆，那么发生故障的可能性会非常高，完全没有安装在坚固外壳中那么耐用。

相比于本章介绍的“梦幻”版本，购买廉价版的 DDS 信号发生器能为你节省大约 30 美元支出。也就是说，只需大约 65 美元就能买到。如果货比三家、谨慎购物，你会花得更少。但如果能制作出更经久耐用又便于携带的信号发生器，那么没有必要为了省下 30 美元而不这样做。但还是那句话，这完全取决于你如何选择。

11.9 成本明细

表 11-6 列出了制作信号发生器时用到的主要元件。注意，总成本中并未包含导线、焊料、束带等材料，当然也不包含运费。有时，一些销售商并不提供低价的 USPS 航运，而坚持使用较贵的 UPS 航运（每 9 美元货品收取 4.95 美元运费）。以后我会尽量避开这些销售商，除非他们采用更合理的运送方式。与此相反，许多中国销售商都是免运费的，但运送周期较长，你可能在几周之后才能收到物品。还是到处逛逛，货比三家吧。eBay 是个不错的起点，如果不喜欢，也可以去 Amazon.com 找类似的商品，通常都是同一家销售商在销售。

正如我在本章开头提到的，制作信号发生器要比直接购买某些现成的信号发生器更贵。不过话又说回来，自己动手制作信号发生器时，你可以进行完全控制，可以让它产生高频输出，这对业余无线电与高速数字应用非常有用。

表11-6：元件价格表

数量	元件	来源	单价	总价
1	外壳，Bud IP-6130	Mouser	25.40	25.40
1	Arduino Uno	Adafruit	24.95	24.95
1	DDS 模块	DealeXtreme	7.99	7.99

（续）

数量	元件	来源	单价	总价
1	原型板	Adafruit	9.95	9.95
1 套	螺丝端子适配器	Seeed Studio	7.50	7.50
1	LCD 显示器，16 × 2	Amazon/Uxcell	4.71	4.71
3	BNC 连接器	All Electronics	1.25	3.75
2	香蕉插座	Amazon	0.67	1.34
3	电位器，10K	Amazon/Amico	1.28	3.84
1	电源开关按钮	All Electronics	1.28	3.84
6	微型按钮开关	All Electronics	0.60	3.60
1	DC 电源插座，筒式	Parts Express	1.98	1.98

* 总价为每个元件单价与数量的乘积。

总成本 = 96.33 美元

请注意，总成本不包含导线、焊料、粘合剂、变压器的费用。写作本书时，表中所列价格都是准确的，但仅供参考。实际购买时，各种元件的价格可能会有一些变动。

如果购买时货比三家、出手谨慎，我认为你可以把总成本降低到 75 美元左右，其中甚至包含所有元件费用。如果手头有大量元件（这些元件可能是你以前大批量购入的）可用，比如连接器、开关、电位器，那么使用它们也可以帮你节省一些费用。最后，如果你恰巧有一个坏掉的旧测试设备，而其中有些元件仍然可用，那么可以把它们拆卸下来，然后用到信号发生器上。

11.10　资源

表 11-7 列出了一些分销商与供应商，制作信号发生器时，我从他们那里购买了所需的元件。（是的，我的确密切关注他们的动向，有时可以抵扣税款。如果你还没有这样做，那么应该好好考虑一下，保存所有收据与包装清单。）

购买任何一个模块或元件时，我们总有多种购买渠道可以选择。表 11-7 列出的这些分销商与供应商是我经常购买电子元件的地方，供你参考，这样可以了解购买电子元件时应该从哪里着手。

表11-7：电子元件购买渠道

分销商/供应商	网址
Adafruit	www.adafruit.com
All Electronics	www.allelectronics.com
Amazon	www.amazon.com
DealeXtreme (DX)	www.dx.com
Mouser Electronics	www.mouser.com
Parts Express	www.parts-express.com
Seeed Studio	www.seeedstudio.com

第 12 章

项目：智能调温器

你之前可能听说过“智能调温器”（我觉得很多人至少在广告中看到过它们），甚至你的房子或公寓里还安装着一台。这些“智能调温器”其实是一种可编程的数字温度控制器，有些允许你通过蓝牙或者其他无线连接方式更改设置，即可以在智能手机或平板电脑上通过相关 App 进行操控。还有一些提供数据收集能力，可以通过无线下载收集到的数据，并通过分析这些数据了解供暖或冷却房间时，何时启动设备才最节能。当然，也有一些其实和老式双金属线圈恒温器差不多，只是用 LCD 显示器取代了老式拨盘与开关旋钮。

> 免责声明：如果你打算制作并使用本章介绍的恒温器，必须自行承担相关风险。虽然它使用低压电路，不会产生触电危险，但仍然有可能造成电力过载，或使温度设置超出设备允许的安全范围（不过大部分系统都内置了安全保护装置），从而损坏你的取暖或冷却设备。注意，针对你的取暖与冷却设备，请使用低电压控制电路，千万不要将自制调温器接到高电压（110 V AC 或更高）电路。包括蒸发冷却器与电暖炉。

12.1 背景

事实上，有多种方法可以对一个老式双金属线圈恒温器（如图 12-1 所示）进行改造。这种设备约有 100 年历史了，采用的线圈由两块金属组成，每块金属热膨胀系数不同，温度变化时，线圈就会逐渐收紧或放松，最终导致触点断开或闭合，从而断开或接通电路。一些恒温器会使用一个小小的封装玻璃管，里面包含两个触点与一个水银珠。水银珠移动时，就会连接两个触点，使电路导通（第 9 章介绍的 KY-017 倾斜传感器模块采用相同原理）。控制动作只有开（on）或关（off）两种状态，再无其他状态。

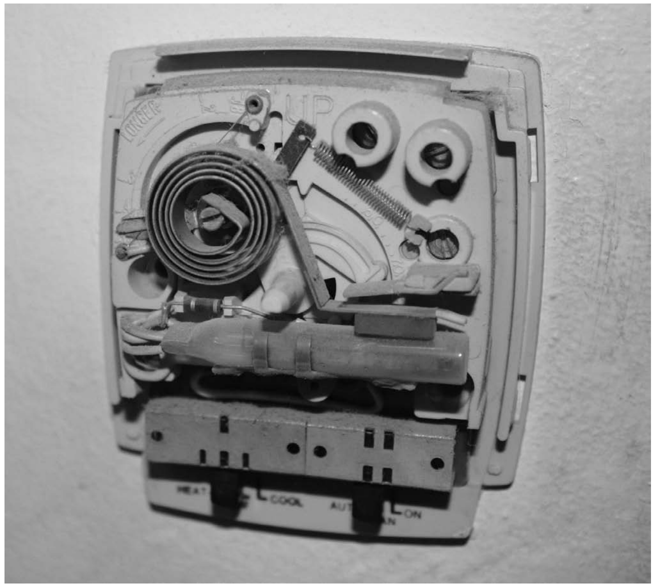


图 12-1：机电式恒温器

本章将使用容易购买的 Arduino 开发板与常用元件制作一个智能调温器，了解在设计、制作、编程过程中都会涉及什么。但在此之前，需要先对要测量什么以及控制什么有个基本的认识。

12.1.1 HVAC概述

调节建筑物内部温度的主要思路是加热与放热。或者换一种说法，变冷的实质是热量散失，通过调节建筑物内部的热量，可以让温度保持恒定。向内部空间释放热量可以通过多种设备实现，比如某种炉子（天然气、丙烷、燃油、木材、煤炭）、电暖炉或太阳能。在热泵系统中，室外空气中的热量会被抽取，并放回建筑中，这类似于反向运转的 A/C 系统（空气调节系统）。为了从建筑物中抽取热量，需要用到制冷系统。其内部装有制冷剂（氨气、氟利昂以及其他任何一种冷媒），通过循环蒸发与冷凝吸走热量。内部空间中的热量会被制冷剂吸收，然后转移到封闭循环系统外部。一些大型冷却系统使用冷冻水实现同样的效果，但这些大型冷却系统的物理体积（包含设备本身及厂房）往往比较大，运行费用高，所以一般只在高校建筑物、高层办公楼、医院以及其他大型设施中才能看到。大型公寓楼通常不会使用冷冻水系统。

许多便民居住设施中，取暖与冷却通常由两套独立的系统负责。但在其他情况下，加热器与空调可能是同一套系统，存在于同一个机柜中，它们多位于房顶或房子周围。事实上，热泵既可以用于加热，也能制冷，这取决于在给定的时间它是如何配置的。热泵通常由两部分组成，一部分位于房内，另一部分位于房外。不管怎样，传统恒温器中通常会有两个开关，其中一个选择自动还是手动操作风机，另一个用于选择模式（加热、冷却或者关

闭）。拨盘或控制杆负责设置目标温度或期望值。整个系统由加热单元、冷却单元与调温器组成，可以看作加热、通风换气、空气调节或者 HVAC 系统。（事实上，提到大型商业系统时，经常会用到 HVAC 这个术语。然而我此处使用它只是为了方便，可以不用写出 heating and air conditioning system 这种完整形式。）除了通风换气功能之外，家用式 HVAC 系统具备大型商业系统的所有功能，只是规模强度要小些。

需要特别注意的一点是，大部分 HVAC 系统都可以而且的确在作为闭环循环系统运行。换言之，它们能持续工作，不断循环建筑物中的空气。虽然有些系统有能力吸入外部空气或者交换内外空气（HVAC 中的 V），但有些系统却不具备这种能力。这样一来，昨天晚饭的焦糊味会停留较长一段时间才会慢慢散去。这也是许多家庭会有独特气味的原因。小时候，我注意到邻居的房子里总是有股炸鸡味，当夏天他们打开 A/C 时就会有这种气味。由此我认为，他们一家人很喜欢吃炸鸡，但我从来没有认真考察过这种特殊气味的真实来源。

12.1.2　温度控制基础

温度控制包含控制子系统，通过加热或冷却操作，使周围环境保持在恒定温度（期望值）。大多数情形下，这是“全开”（all-on）或“全关”（all-off）类型的操作。大多数民用 HVAC 系统中没有中间加热或冷却速率。此处之所以说“大多数”是因为，可能有些人或某个地方安装了可变输出电阻式加热系统（variable-output resistive heater system），但迄今为止，我尚未在实验室或工业环境中看到过类似的设备。

HVAC 系统的“全有或全无”（all-or-nothing）性质意味着，实际的室内温度永远不会准确地停留在设定值上，而只会短时间内在设定值上下浮动。比如，假设我们现在正开着一台空调，目标温度设定为 72 ℉（22 ℃），如果将一天中的室内室外温度记录下来并制成图表，将如图 12-2 所示（图中数据并非真实数据，仅用作演示说明）。

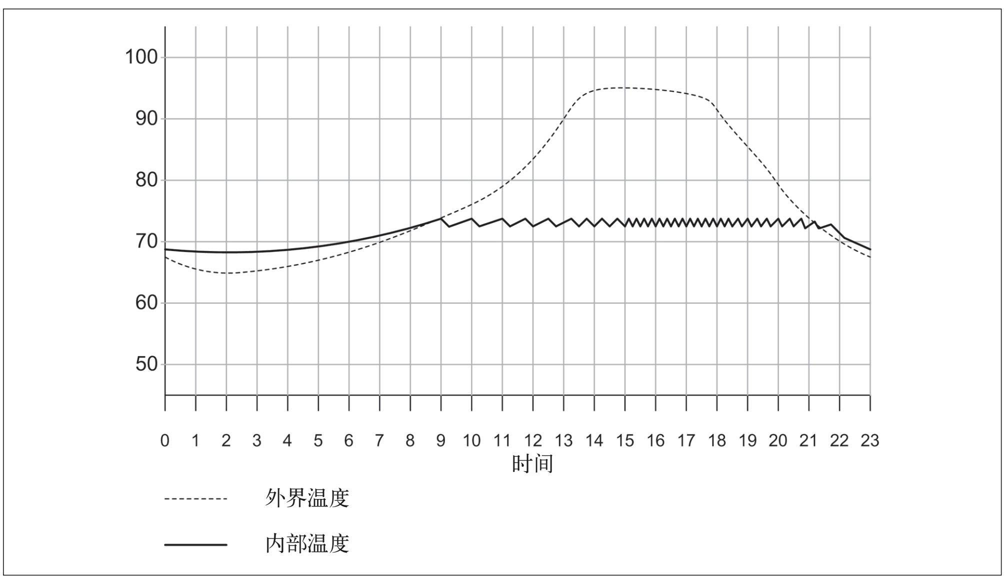


图 12-2：一天中的温度变动图

从图 12-2 中可以看到空调是何时启动的，因为空调启动后室温下降，明显低于室外温度。当天早些时候的循环（打开和关闭）发生得不如晚些时候那么频繁。外部热量要花一些时间才能进入室内，室内外绝缘性越好，室内暖起来就需要越多时间。下午晚些时候，空调努力地将室内温度维持在指定的水平。直到晚上稍晚的时候，随着室外温度下降，工作才开始放缓。

我们把只有“开”和“关”两种状态的控制器称为“滞环控制器”（**hysteresis controller**），也称为“砰砰控制器”（bang-bang controller）。控制器的滞环宽度（amount of hysteresis）决定着冷却或加热设备的启动频率以及工作时长。你可以将滞环想象成系统开启与关闭之间的时间间隔，它基于两者之间的设定值。与此对应，一个物理上的例子是三孔活页夹的开合动作。打开扣环后将其再次合上需要付出些努力，而一旦打开或合上扣环，这些扣环将一直保持当前状态不变。

图 12-2 的数据是虚构的，但仍然能够如实反映出，典型住宅中的传统调温器对温度的调节作用。当内部温度约为 73 ℉时，空调上电运行，期间一直处于运转状态，直到温度下降到 71 ℉以下。这意味着该假想系统滞后大约 2°，这个范围称为“滞环宽度”（其实，该系统中滞环宽度太窄，导致当天晚些时候 A/C 循环过于频繁）。

HVAC 系统的每次开 / 关（on/off）动作都被称为一个循环。图 12-3 描述了开关点之间的差值变小（即 Hmax 与 Hmin 之间距离变小，该距离被定义为滞环宽度）时会发生什么。滞环宽度变窄时，系统能够更好地将温度维持在指定水平，但会造成电源开关过于频繁。我们希望将滞环宽度设得尽可能宽，因为加热器或 A/C 设备每次循环都会减少其使用寿命，并且使电费增加。

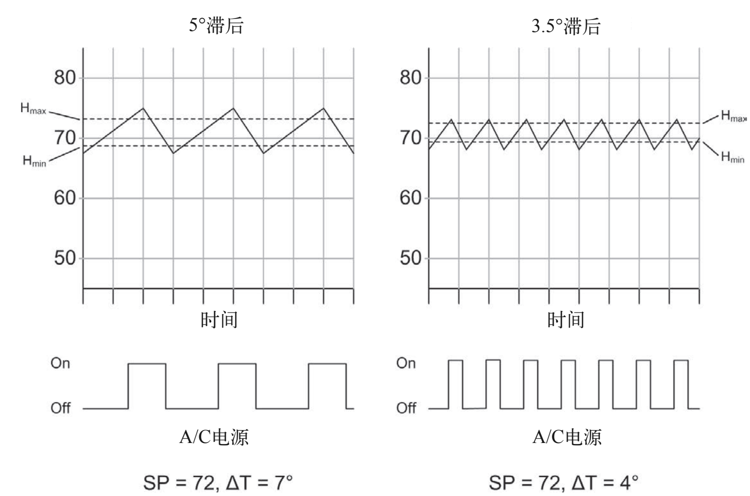


图 12-3：滞环效应

虽然 5°滞环宽度更节能，并且不会像滞环宽度为 3.5°时那么频繁循环而磨损设备，但这也意味着实际温度在冷热之间 7°范围内波动（请记住，图表中的数据并非真实数据，仅用作演示说明）。

有多种因素可以影响加热或冷却系统的循环频率（开关频率），其中最明显的是 A/C 运行时热量散失的速度，或者加热器运行时热量产生的速度。另一个不太明显但同样重要的是，湿度与室内空气流动。随着湿度增加，空气能够保留更多热量，这在加热时很有帮助。A/C 设备也可以充当除湿机，当空气经过系统中的冷蒸发器线圈时，其中的水分就会被冷凝排走，干燥空气更容易蒸发汗液，这有利于快速降低我们的体温。静止空气（也就是说，空气不会流动，除非打开 HVAC 系统）容易在建筑物中造成热点区域（hot spots）或阴冷区域（cold spots），如果其余空气不流经调温器，那么调温器最终只能对一部分内部空间进行调整。

理想情况下，温度控制系统应该能够感知建筑物所有部分的温度，但现实情况往往并非如此。现实世界中总有些房间的某些区域在白天大部分时间受到阳光直射，而其他区域处于阴影中，或者有些房间空气流动不畅。这样，即使调温器努力工作，使其周围区域的温度维持在指定水平，但建筑物的其他部分还是不能被准确地加热或冷却。只要让周围的空气流动起来，就可以很好地解决这一问题，此外，监控空气湿度也有利于判断何时打开风机或何时启动加热与冷却系统。相比于运行加热器或 A/C 设备，单独运行风机会便宜很多。

12.1.3 智能温度控制

智能调温器背后的主要目的是减少能源浪费，以及在建筑物内实现更均匀的温度分布。为此，要恰当地调节循环时间（比如调节滞环宽度），仅当房间有人时才启动系统，根据一天中的时间或一周中特定的一天更改设定值，以及充分利用 HVAC 系统中内置的风机。

现代商业数字恒温器有一些共同特征，比如可调的循环时间、自动切换加热 / 冷却模式，以及逐日可编程能力（day-by-day programmability）。下面简单了解一下这些能力，并分析哪些对本章项目有意义。

“循环时间可调”使得我们可以减少开关加热或冷却系统的频率。只有温度达到指定值或在其附近时，才会出现开关加热或冷却系统的情况。即温度变化很剧烈时，才会打开加热器或空调（A/C）。其实，这一过程中需要做的是增加系统的滞环宽度，这样温度在指定值附近浮动时，循环周期会更长。相反，如果想让系统对温度在短时间内的变化做出迅速响应，则应该减少滞环宽度。

“自动切换”（auto-changeover）允许系统根据需要在冷却与加热两种操作间自动切换。比如，白天很热而晚上很冷时，到了夜晚，调温器会自动从冷却切换为加热，以保证室内温度相对恒定。我们要制作的 Arudino 调温器支持“自动切换”与“循环周期可调”功能。

智能调温器的一大特征是“计划调度”（Profile scheduling）。大部分可编程数字调温器都有能力根据调度计划，在一周中指定的日子启动加热或冷却系统。当你平时去上班而家里没人时，加热器或 A/C（空调）将不会工作，这样能节省一些电费。同样地，有些机型允许家里人晚上都睡着后自动调低加热或冷却程度。针对白天、晚上、周末时段的运行，我们的控制器也支持创建计划配置的功能。

数字调温器封装于一个好看的外壳中，带有高科技发光面板，磨砂塑料背后是数字显示器，但这些都是营销噱头。其实，数字调温器也可以放置在普通塑料外壳中。虽然看上去可能没有商业产品那么有科技感，但这一点都不影响正常工作及其易用性。通过为本章项

目使用廉价外壳，你可以省钱省时。我决定使用的肯定是低成本的。

12.2 项目目标

本章项目的主要目标是创建一个数字调温器，用于替换老式家用调温器。从网上可以找到许多基于 Arduino 的恒温器项目的资料，本章项目与其中很多项目类似。只有几种方法可以将 Arduino、温度传感器、简单显示器、一两个继电器攒在一起。本章项目的独特之处是，它集成了一个湿度传感器，并具备单独使用风机的能力，以便让周围空气流动，在不启动加热器或 A/C 设备的前提下，将冷暖空气转移到需要的地方。

我们的 Arduino 调温器也有能力使用内部温度与湿度数据，借以判断是否需要调整循环时间。此外，它也提供了更多扩展，允许用户添加其他传感器，用于感知外部温度与湿度。最后，它使用继电器连接到现有的 24 VAC 控制电路。美国大部分家用 HVAC 系统都使用这种控制电路，所以不存在严重的触电危险，也不会对你的加热或冷却设备造成实际损害（至少在电气结构上不会，但仍然有可能会因为循环过快而导致压缩机或加热器点火部件损坏）。

本章项目非常简单，只需要少量元件以及很少的焊接工作。事实上，两个主要挑战是为设备选择一个好的外壳，以及编写软件。

我们的 Arduino 调温器的设计目的在于替换一个老式低电压四线制民用调温器，但不局限于四线制系统。其实，控制一个暖炉或 A/C 设备是很简单的事，只要对软件做些微小的改动即可，也不需要使用所有可用的控制输出。单独控制一个加热器或者空调设备很简单，只要不使用所有可用的控制输出，以及对软件做少许修改即可实现。

12.3 定义与规划

根据前面介绍的内容，可以确定要集成到设计中的基本功能。其中一些是老式四线制调温器中已经有的，另一些是基于湿度新添加的：

- 实时时钟
- 内部湿度传感器
- 内部温度传感器
- 自动加热或冷却操作
- 自动风机控制
- 七日运行计划

从许多方面来说，基础版的 HVAC 控制器与家居建材大卖场（还有一些本地的硬件小卖场）、HVAC 售卖处、各类在线电子产品供应商销售的那些类似。

本项目有意回避直接使用 HVAC 系统中的高电压 AC 控制电路，仅打算用在低电压（24 VAC）系统中。高电压 AC 可能损坏你的 HVAC 设备，烧掉你的房子，甚至杀死你（不一定按此顺序）。如果你需要那种系统，请考虑使用达到 UL 与 CSA 安全等级的商业控制器，并且雇用一位专业电气工程师或 HVAC 技术人员帮你安装。

我们要制作的控制器将使用 Arduino Nano，Arduino Nano 安装于螺丝端子台原型板上。实时时钟（RTC）模块记录日期与时间。Arduino PCB 堆叠板安装在外壳盖上。四芯继电器模块安装在外壳里面，位于底部面板。所有连线经过底部的一个小孔进入外壳。

12.4 设计

Arduino 调温器用于替代标准的四线制调温器，它并非为多级式加热 / 冷却或热泵系统设计。因此，最适合那些使用旧式调温器（如图 12-1 所示）的老房子。

12.4.1 功能

Arduino 调温器有 3 种基本功能：加热、冷却、换气。图 12-4 展示了主要组成部分的框图。成功的秘诀是，如何使用这些基本功能实现高效运行。

旋转编码器用途广泛，比如设置温度、调整日期等。LCD 扩展板包含一系列按键，但本项目不会使用它们。

许多老式 HVAC 系统只有 4 根连线，一些新系统还有备用的电源线。如果 AC 电源可用，并且外壳中也装有小型电源，那么我们的 Arduino 调温器可以使用它。在我们设计的调温器中，我将使用一个外部的壁式电源或墙疣。最理想的情况是，使用来自 HVAC 的电力，但许多老式系统没有备用的 24 VAC 电源线。

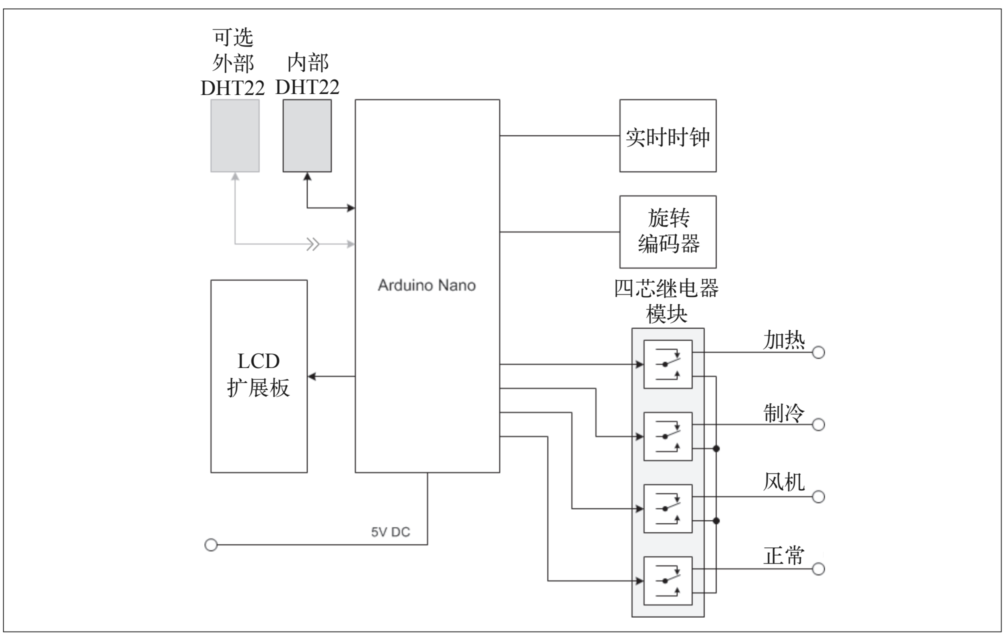


图 12-4：Arduino 调温器框图

图 12-5 是标准老式调温器常见的内部接线图。当然，具体的内部接线细节根据品牌不同而

各不相同，但基本思路是一样的。

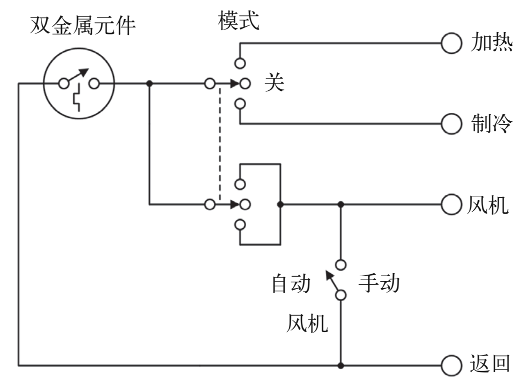


图 12-5：标准老式调温器内部电路

对于这种老式调温器，需要特别注意风机接线，这使得它无论在加热还是冷却时都会通电运转。风机不运行的情形下，并不应当运行 A/C 或加热器。在一些系统中，加热器风扇独立于 A/C 风扇，仅当加热器中的内部温度达到指定值时才会启动运行。

12.4.2　外壳

针对本项目，我选择了一个塑料的电气接线盒。它带有一个可拆装的盖子，适合安装到墙上，如图 12-6 所示。进行一些必要的处理之后，我们将为盒子刷上中性色，让它显得更好看。



图 12-6：Arduino 调温器外壳

如你所见，这个外壳真的很丑，对此我不否认。但对于本章项目而言，我们主要关心的是

它是否可以安装到墙上，以及是否有足够的内部空间安装各种电子元件。我们会把侧面的安装突耳移除，并对前盖进行打磨抛光，让它看上去更美观。

控制部分很简单，只有一个 LCD 显示器与一个旋转编码器，它们都安装在前盖板上。前面板布局设计如图 12-8 所示。

继电器模块安装在外壳底部，调温器的现有连线通过盒子底部的空洞进入外壳内部。RTC 模块安装在外壳内部，DHT22 附着到外壳底部，以便可以暴露在周围环境中。之所以这样安装 DHT22 而不将其安装到外壳内部是因为，我不想在外壳上钻一排孔供空气流动。另一种方法是在外壳上开一个方孔，使其恰好可以容得下 DHT22，这样就可以把 DHT22 安装到外壳内部，同时保证它能接触到外部空气。4 枚螺丝将调温器固定到墙上。

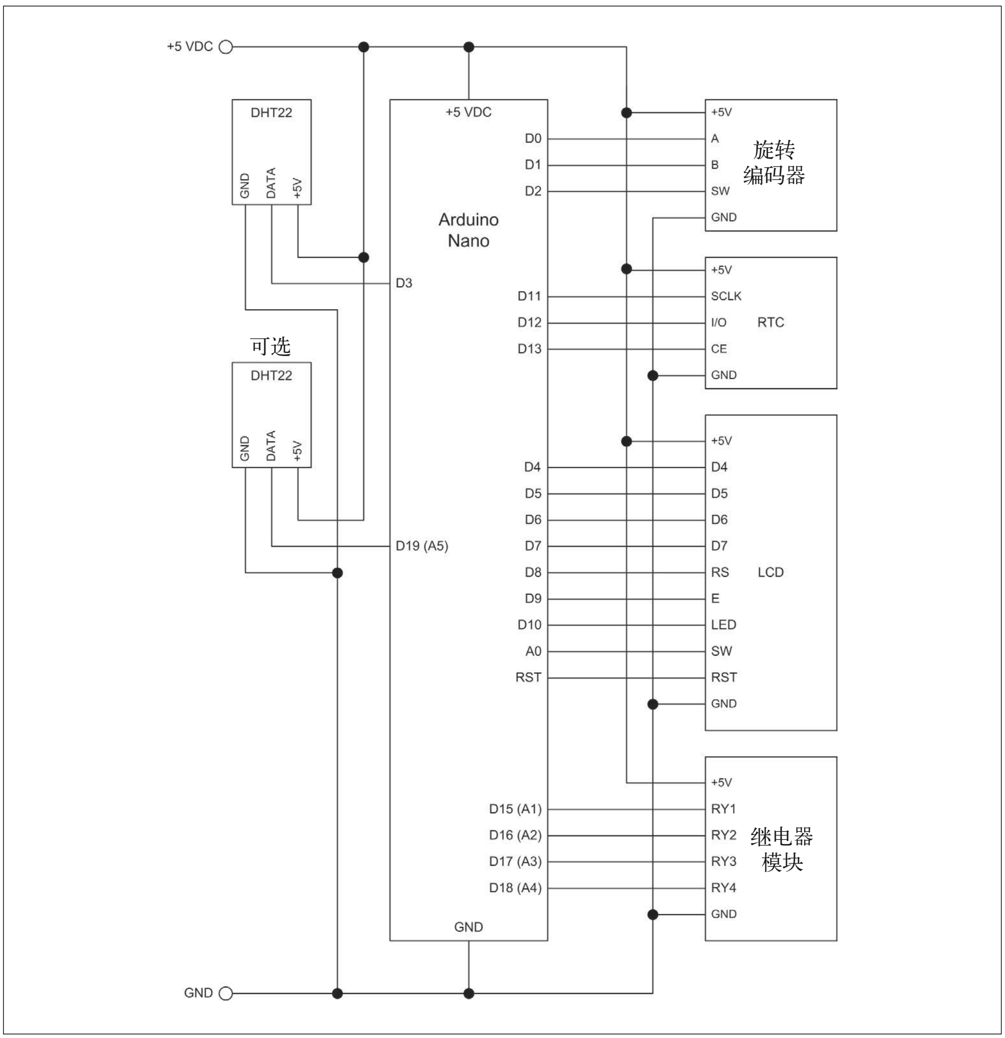


图 12-7：Arduino 调温器电路图

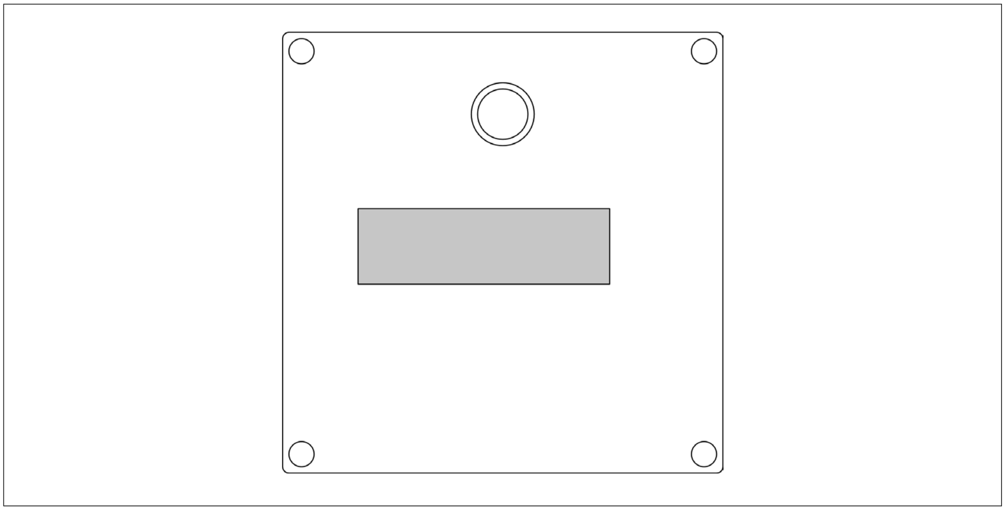


图 12-8：Arduino 调温器前面板布局

12.4.3　电路图

原型与最终产品之间没有明显的差别，所以只有一版电路图。从电气结构上看，原型与最终调温器是一样的，但原型使用的是 Arduino Uno，而不是 Nano 开发板（两者 AVR MCU 是一样的），主要因为原型夹具上安装的就是 Arduino Uno。

请注意，MCU 上每个单独的引脚都会用到。旋转编码器通过 D0 与 D1 共享 Arduino 编程接口。只要不用旋转编码器，就不会产生冲突。

此外，你可能也注意到，模拟输入引脚被当作数字 I/O 引脚使用。这完全正常，回顾第 2 章与第 3 章就会发现，ATmega328 的模拟输入端口（C 端口）也是标准的数字 I/O 端口。通过引用 D14 与 D19 引脚，可以像使用其他数字 I/O 引脚一样使用这些模拟输入引脚。当然，现在模拟输入还不可用，但由于调温器不使用任何模拟输入（除了 A0 引脚，LCD 扩展板会使用它），所以这不会有什么问题。

我们不能因为 Arduino 官方将某些功能（比如模拟输入功能）指派给特定引脚，就认为这些引脚只有这些用途。这只是 Arduino 官方对它们的命名而已。Arduino 开发板只是 MCU 的一个“包装”，在板子引脚与 MCU IC 之间并没有做什么特别处理。真正决定一个引脚能做什么或不能做什么的是 MCU，而不是印在 PCB 上的标签。

12.4.4　软件

本项目是软件密集型（software-intensive）的，但软件的大部分则关于用户界面与情景模式配置。调温器运行的实际控制逻辑并没有那么复杂。

请记住，本书讲解的重点是 Arduino 硬件及其相关模块、传感器、组件，书中给出的示例代码都是软件中最关键的部分，并不是完整可运行的代码。关于示例与项目的完整软件代码，请前往 GitHub（https://www.github.com/ardnut）进行下载。

每次运行主循环，软件都会检查是否有来自旋转编码器的用户输入，更新当前温度与湿度数据，判断显示是否应该在各屏之间进行切换——前往各种设置界面还是修改值的界面。中断处理程序（Interrupt handlers）捕获来自旋转编码器的用户输入。高级版本软件框图如图 12-9 所示。

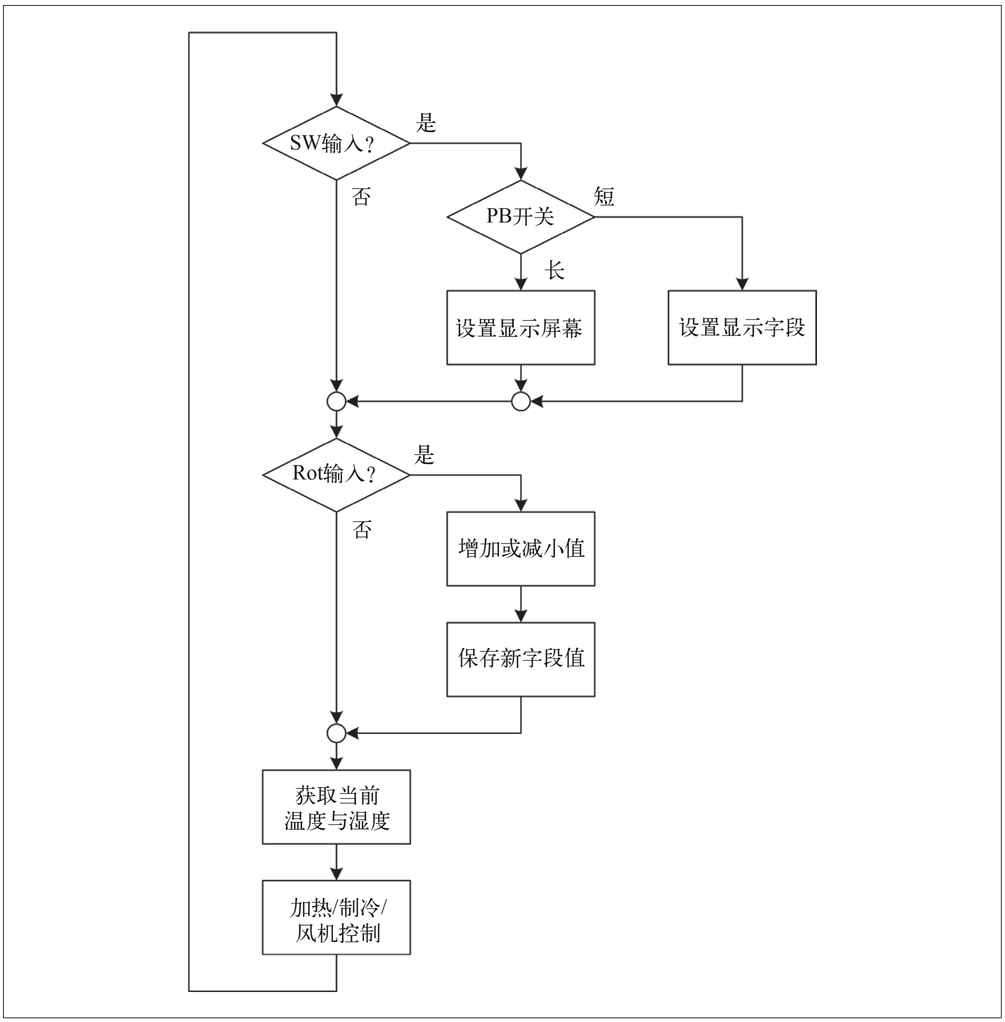


图 12-9：Arduino 调温器软件框图

图 12-9 中的 Arduino 调温器软件框图描述了实际源代码主要做了些什么。除了 Thermostat.

ino 主文件之外，软件还包含用中断处理用户输入、管理计划调度（白天、夜晚、周末）、更新显示内容的模块。

软件首先要做的是，检查旋转编码器上的开关（SW），通过读取开关信号可以了解用户想做什么。旋转编码器中的按键开关在两个屏幕或者一个屏幕的两个区域之间进行切换，至于究竟是哪个动作，则取决于开关按下时间的长短。

旋转编码器开关与 A/B 输入的中断处理程序会在全局变量模块中设置标志。主循环会检查这些标志，以判断采取哪种显示更新动作（如有）。中断处理程序并未在图 12-9 中显示，请参考 12.6 节。

软件框图中，“加热 / 冷却 / 换气控制”部分是调温器真实的控制逻辑。如前所述，这是一个离散态的滞环控制器，也叫“砰砰控制器”。图 12-10 描述了一个 A/C 系统中温度、时间、滞环宽度之间的关系。对于加热系统，只要简单反转操作即可。

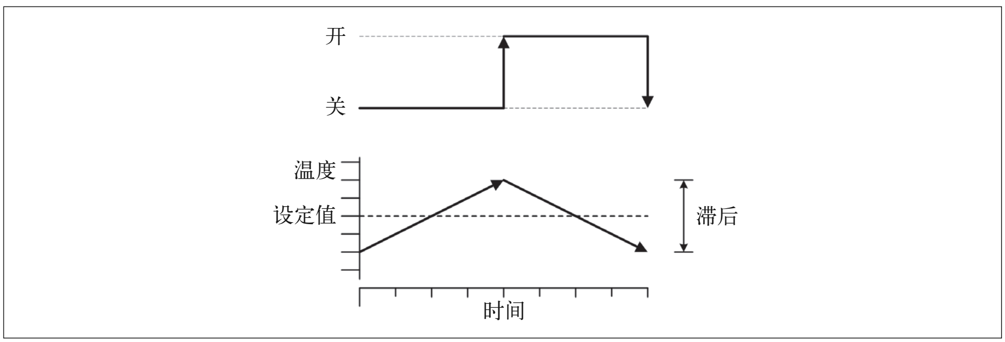


图 12-10：时间、温度、滞环宽度

图 12-11 是对这种关系的另一种描述。图 12-10 中的折线表明，滞环宽度是温度的函数。两种图传递的信息是一样的：带有 on/off 驱动器（加热器或 A/C 设备）的“砰砰控制系统”（比如调温器）无法将温度准确控制在指定的数值上，最后得到的温度总是在滞环宽度的最高点与最低点之间浮动。

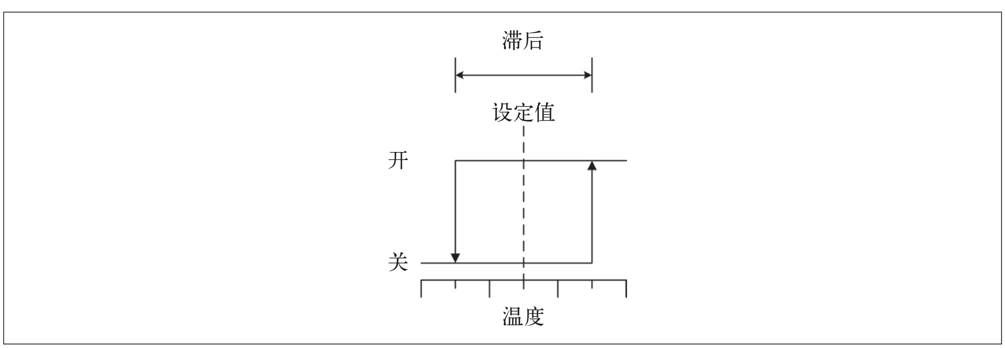


图 12-11：控制系统响应的滞后效应

在真实系统中，温度上升或下降的速度取决于热量进入或排出系统的速度，以及加热或冷

却设备加热或抽出热量的效率。最后结果是，加热与冷却次数几乎从来不一样——虽然图 12-10 与图 12-11 中它们可能看上去是一样的。

图 12-12 显示的是调温器的控制逻辑。请注意，加热与冷却是一对相反的操作。在老的机电式调温器中，H（hysteresis）值是通过双金属线圈上的定位螺丝确定的。而在软件中，我们可以动态设置，只要你想。

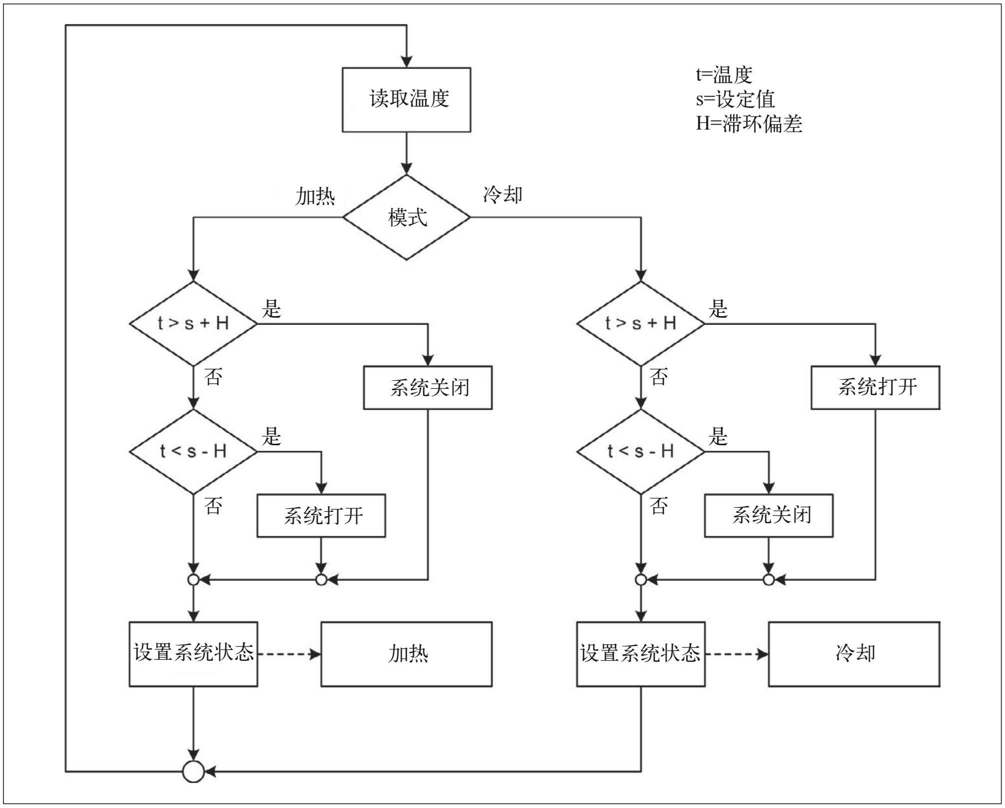


图 12-12：调温器的加热与冷却控制逻辑

12.4.5 用户输入/输出

控制输入由单个旋转编码器组成。这看上去有点怪，但其实编码器上也集成了一个开关，用户按旋钮时，开关即被按下。开关在显示配置（有时我把它们叫作“屏幕”）之间进行切换。每个屏幕包含若干个域，用于显示数据或设置。旋转编码器也可以用来在同一个屏的不同域之间进行移动。旋转编码器按键开关被按下的时间长短决定软件采取哪种动作，短按表示选择，长按表示切换到下一屏。

本章项目使用的是一块 16×2 LCD 显示器，它与第 11 章信号发生器中使用的那块一样，并且已经预装在了 PCB 上，这极大地简化了内部连线。LCD 背光通常是关闭的，除非有控制输入发生。我们的项目不会使用 LCD 扩展板上的小型按钮，但只要你愿意在前面板

钻很多孔并找到某种用户可按的扩展件，即可使用它们。此处我选择不使用。

通常情况下，控制输入未被激活时，两个主要屏幕会交替显示。第一屏显示当前温度读数、期望值、操作模式与情景设置（若有），第二屏显示当前日期与时间。图 12-13 给出正常运行时屏幕上的显示情况。符号 ^、v、– 分别指示相对于前一个读数，当前读数正在上升、下降或稳定不变。

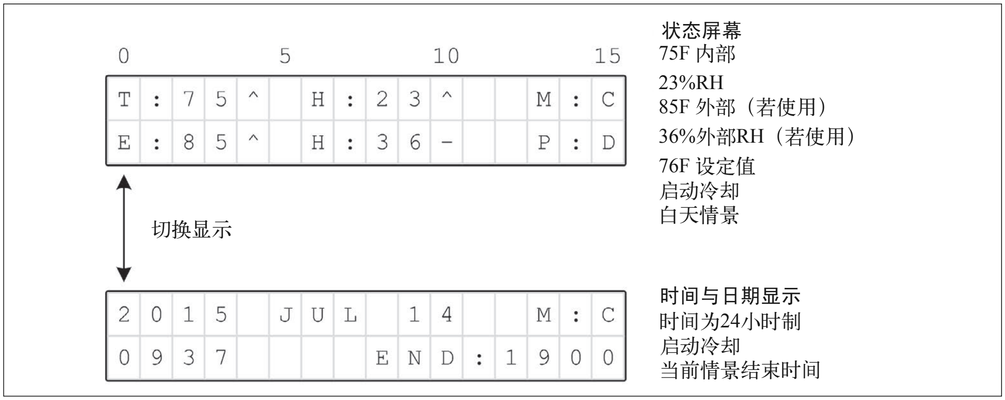


图 12-13：正常运行时的屏幕显示

表 12-1 定义了图 12-13 显示的各个域。第二屏中，END 域指示当前情景设置结束的时间。

表12-1：屏幕上各个域的定义

域	用途
T	当前内部温度数（℉或℃）
E	外部温度（若启用）
H	湿度
M	模式：A（自动）H（加热）、C（冷却）、F（风机）、X（关闭）
P	情景：D（白天）、N（夜间）、W（周末）、X（无）

图 12-13 屏幕上显示的是纯信息，无可修改域。温度单位可以是℃，也可以是℉，默认使用℉。显示时间时，我选用了 24 小时制格式，你可以很容易地把它改成 12 小时制（AM/PM）。

图 12-14 显示的是设置界面，通过它，用户可以打开或显示情景配置。如果情景配置不可用，用户可以手工调节温度期望值、温度范围、滞环宽度、运行模式。

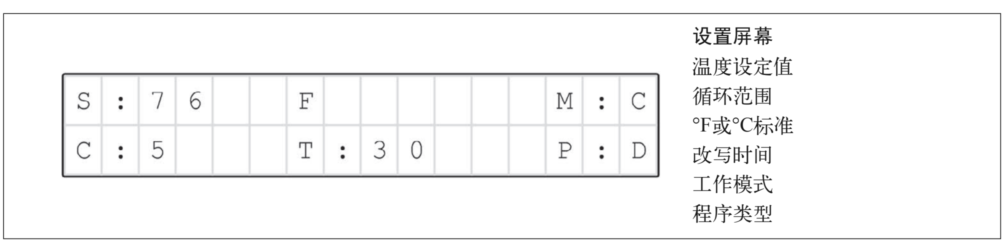


图 12-14：设置界面

设置界面是控制调温器的主要场所。用户只有通过特定操作才能将其调出并显示。设置界面中各个域的定义如表 12-2 所示。

表12-2：设置界面中各个域的定义

域	用途
S	期望温度值
C	循环范围（滞环宽度，度数）
F	℉或℃温标
O	重写时间
M	模式：A（自动）、H（加热）、C（冷却）、F（风机）、X（关闭）
P	情景设置：D（白天）、N（夜间）、W（周末）、X（无）

调温器主要的预置控制格式是用户定义的情景设置。图 12-15 显示的是情景设置编辑界面。

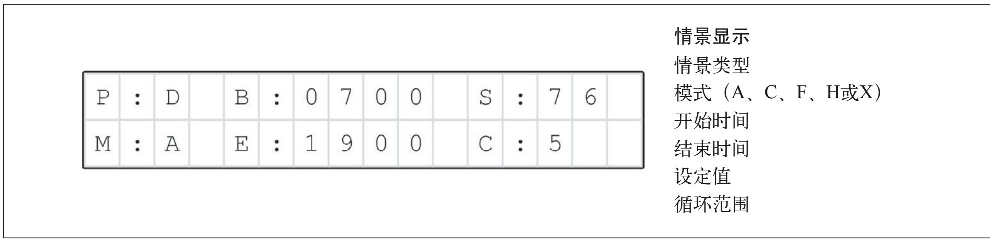


图 12-15：情景设置编辑界面

调温器共有 3 种情景设置，分别为白天、夜间、周末，对应的标识依次为 D、N、W。情景设置域处于激活的情形下，转动旋转编码器的转轴将依次在 3 种情景之间切换。每种情景模式都有开始时间与结束时间。就“周末”情景来说，默认起始时间为周五午夜 0 点，结束时间是周日午夜 0 点。如果一个情景的结束时间与另一个情景的开始时间重叠，那么前一个情景的结束时间优先。情景编辑界面中各个域的定义如表 12-3 所示。

表12-3：情景编辑界面中各个域的定义

域	用途
P	情景：D、N、W
B	开始时间
E	结束时间
S	温度期望值
C	循环范围（滞环宽度）
M	模式：A（自动）、H（加热）、C（冷却）、F（风机）、X（关闭）

在各个屏幕与域之间进行导航确实需要我们做些研究学习。通过按压旋转编码器开关的长短与使用树状结构，可以让用户顺利在不同屏幕与域之间进行导航切换，不致于迷失。图 12-16 显示了各个屏幕与域是如何响应按钮的长按与短按动作的。

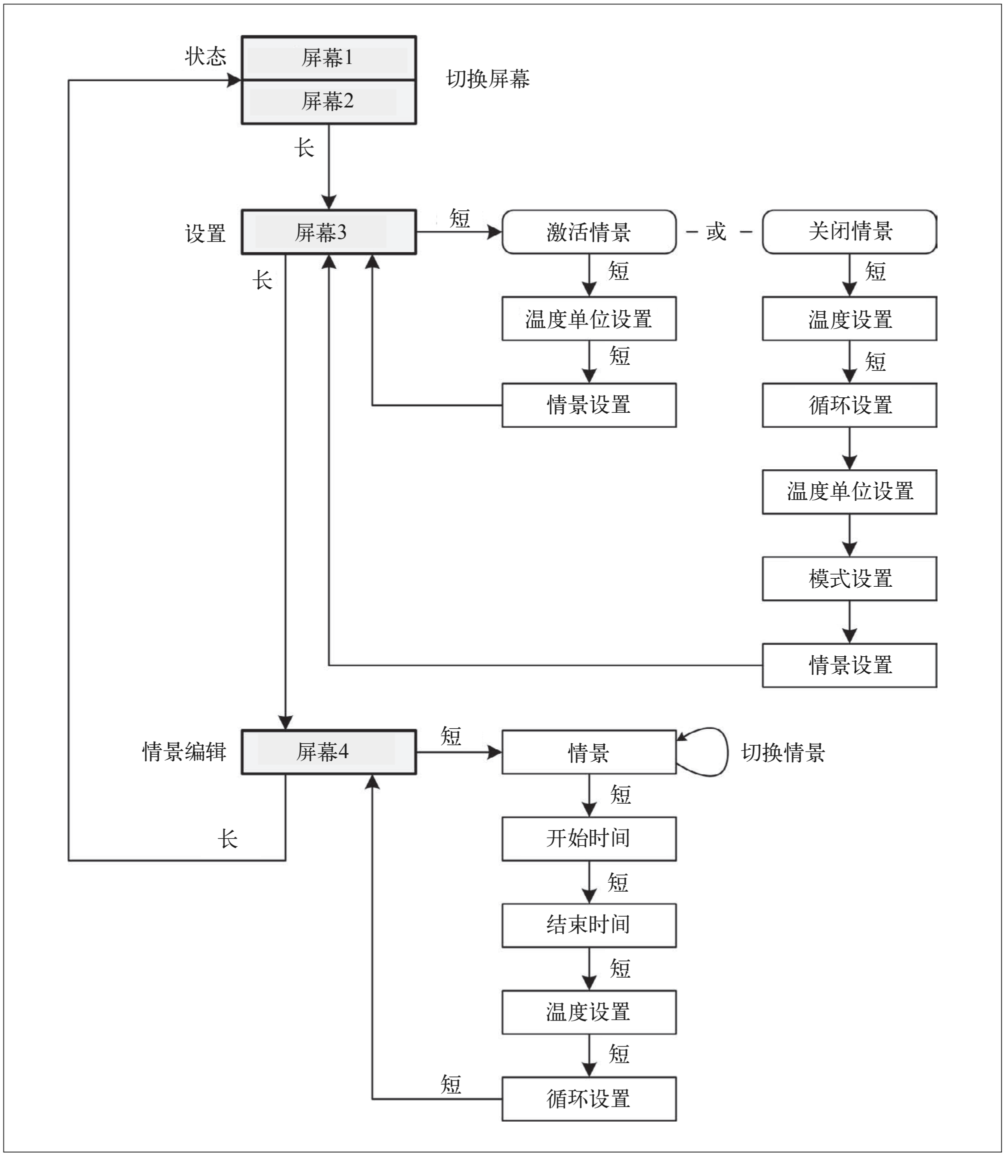


图 12-16：在调温器屏幕与域之间导航

图 12-17 显示了调温器中用到的 4 个屏幕，这使安排设置这些屏幕变得更容易。

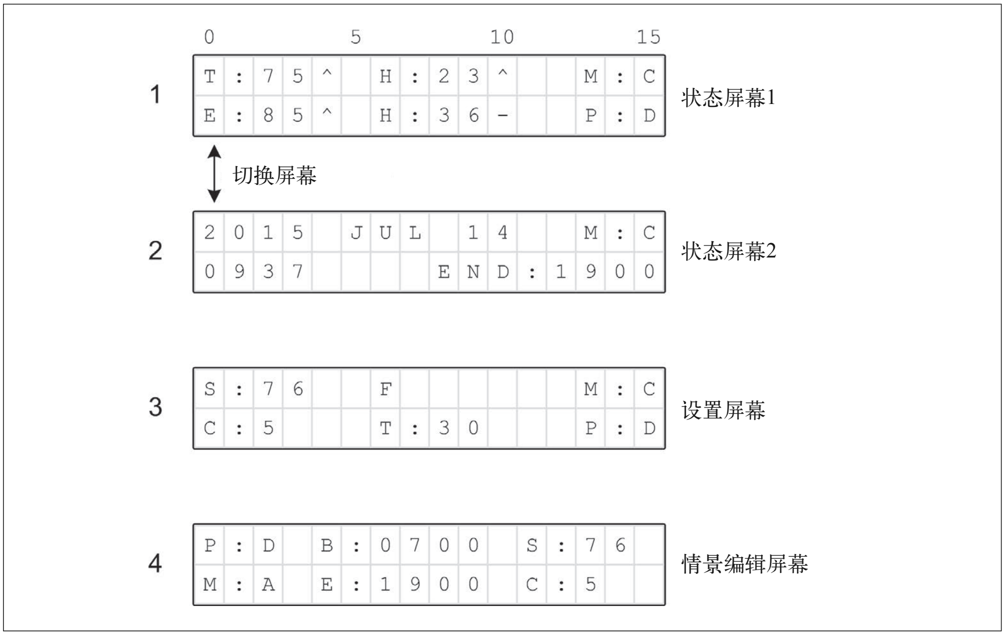


图 12-17：调温器使用的 4 种屏幕

按下按钮超过 1 s 时为“长按”，短于 500 ms 时为“短按”。软件使用中断与计数器监视按钮按下的时间。

12.4.6 控制输出

控制输出由一个四芯继电器模块组成，它带有 4 个独立的 SPDT 继电器，并且集成了继电器驱动器与 PCB 端子台，所以很容易将其与现有的调温器线缆连接起来，如图 12-18 所示。公共线经过第四个继电器（图中右下角）。

图 12-4 所示的框图中，源于现有 HVAC 系统的公共线也经过其中一个继电器。这是一个故障安全措施。如果调温器失去电力，连接公共线的继电器将会断开，HVAC 其他功能不会被打开。所有外部 HVAC 控制电压线（包含公共回路）都连接到继电器的 C（公共）与 NO（常开）端子。

12.5 原型

本项目的原型由 Arduino Uno、一对接线端子扩展件、LCD 扩展板组成。它们基本上与最后安装在外壳中的那些元件是一样的（Nano 与 Uno 都使用 ATmega328 MCU）。

我重用了板载 Arduino，第 11 章制作信号发生器时我们用过。当然，也可以使用第 10 章介绍的 Duinokit。我选用这个开发板来做另外一项任务，因为它很小，测试期间可以很方便地安装到现有调温器附近的墙面上。原型如图 12-18 所示。

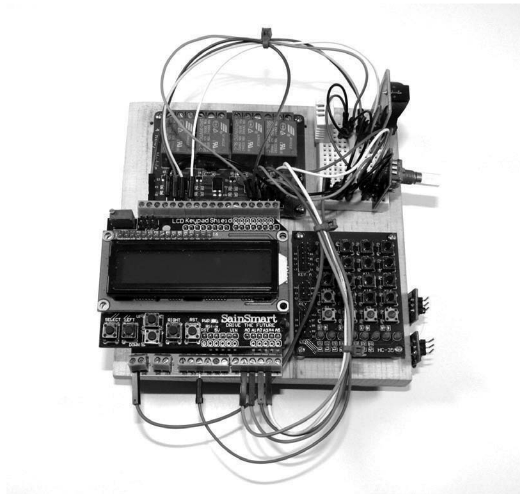


图 12-18：Arduino 调温器原型

原型中使用的元件与最终产品稍微有些不同，但在电气上是等效的。LCD 扩展板与最终产品中使用的一样。原型中使用的是 Arduino Uno，而最终产品中使用的是 Arduino Nano。在图 12-18 中可以看到，位于小型免焊面包板模块上的旋转编码器、RTC 模块、DHT22 传感器。原型中用到的各种元件如表 12-4 所示。

表12-4：原型调温器元件列表

数量	元件
1	Arduino Uno（或类似设备）
1	16 × 2 LCD 显示器扩展板
1	DS1302 实时时钟模块
1 套	螺丝端子适配器
1	四芯继电器模块
1	DHT22 温度 / 湿度传感器
1	KEYES KY-040 旋转编码器模块

12.5.1　DHT22传感器

我们将使用 DHT22 传感器设备感知环境温度与湿度，第 10 章的 GreenShield 中也使用了这种传感器。在最终成品中，我们会将引脚插入螺丝式端子台，从而将 DHT22 传感器连接到 Arduino Nano。图 12-19 显示了 DHT22 传感器的各个引脚。

你可以从 Adafruit（http://www.adafruit.com）、SparkFun（http://www.sparkfun.com）或其他地方获取 DHT22 传感器的数据手册，该手册建议用户在数据引脚上连接一个 1 KΩ 的上拉电阻。在原型中，借助木板基座上一个小的免焊面包板将 DHT22 连接到系统，如图 12-18 所示。

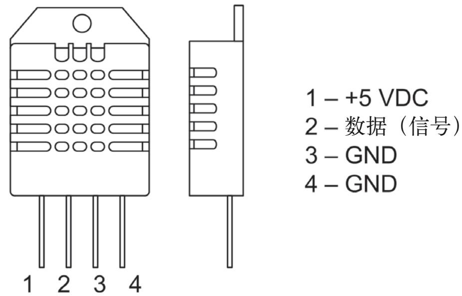


图 12-19：DHT22 传感器引脚

12.5.2 旋转编码器

调温器的主控制输入（也是唯一的控制输入）是一个 KEYES KY-040 模块（关于 KEYES 模块的更多信息，请阅读第 9 章相关内容），它带有一个 Bonatech 旋转编码器，类似于 Alps EC11 元件。图 12-20 显示的是 KEYES KY-040 模块，其引脚见图 12-21。

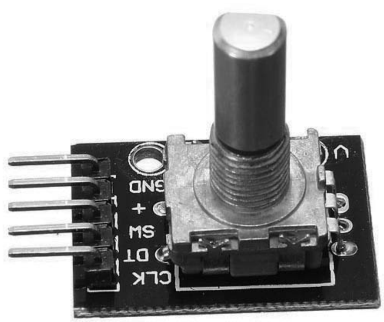


图 12-20：旋转编码器模块

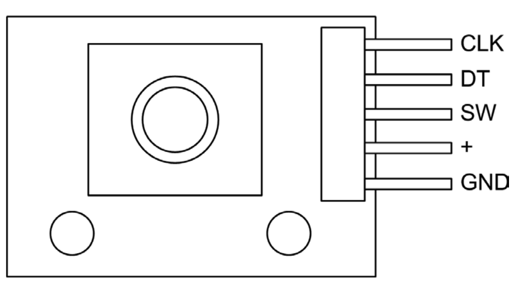


图 12-21：旋转编码器模块的各个引脚

旋转编码器模块电路简单，如图 12-22 所示。编码器内部包含一个开关，按下转轴时即触发开关，3 根信号线使用 10K 上拉电阻。带有 CLK 与 DT 标签的引脚依次对应于 A 与 B 信号，这在大多数软件示例与简单的正交旋转编码器中很常见。

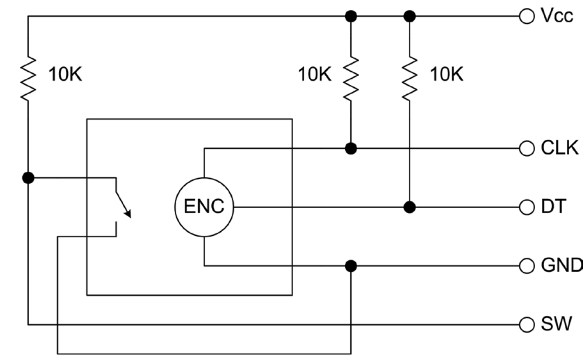


图 12-22：旋转编码器模块电路图

旋转编码器的集成开关在各个显示屏幕之间进行切换，也可以设置某个显示屏上的各种可修改项。集成开关和旋转编码器自身是唯一两个控制输入，内部集成开关是常开的。

12.5.3　实时时钟模块

调温器要有计时的能力，以判断要使用哪种情景设置。为此，我们将使用一个实时时钟模块（RTC），它基于 DS1302 IC 实现。实时时钟模块其实只是个载体，具体由 DS1302 IC、晶振、电池组成。图 12-23 显示了一个 RTC 模块的电路图，其引脚一目了然。

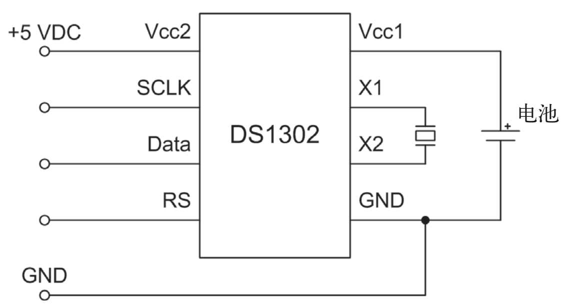


图 12-23：RTC 模块电路

可以从 Arduino Playground（http://bit.ly/apgds1302rtc）下载 DS1302 库使用。我们的调温器项目将直接使用这个库，不会重新编写新库。

12.5.4 LCD扩展板

调温器项目中使用的 LCD 扩展板很常见，你可以从多种渠道购买，如图 12-18 所示，相关介绍请参考第 8 章中的有关内容。LCD 扩展板是 Nano 数字 I/O 引脚的最大"消费者"。我们选用的 LCD 扩展板集成了一个电位器，用于调节对比度；还集成了一个控制晶体管，用于控制显示器背光。LCD 扩展板可以直接插到底部的 Arduino 上，这样可以省去很多连线，不像第 11 章制作信号发生器时那么凌乱。

12.6 软件

调温器的软件很简单，类似于大部分嵌入式软件，包含一个主循环。启动后，这个主循环会不间断运行。与目前为止我们见到的其他示例不同，调温器利用中断处理旋转编码器的输入。在 Arduino 上实现中断处理程序不是特别难，到目前为止，中断是处理外部世界异步事件的最好方式。

12.6.1 源代码组织

与本书中的其他项目一样，源代码被组织成一系列源文件（见表 12-5）。其中包含主模块（含有 `setup()` 与 `loop()` 函数）、全局变量模块、控制 HVAC 功能的逻辑模块（根据用户定义的加热与冷却程序）。显示模块借用的是第 11 章中为信号发生器编写的模块，接口模块（tstat_iface.cpp）包含旋转编码器的中断处理程序。

表12-5：调温器各源代码模块

模块	功能
Thermostat.ino	主模块，包含 `setup()` 与 `loop()` 两个函数
tstat.h	常量定义（`#define` 语句）
tstat_ctrl.cpp	HVAC 控制逻辑
tstat_ctrl.h	包含文件
tstat_gv.cpp	全局变量
tstat_gv.h	包含文件
tstat_iface.cpp	控制输入处理
tstat_iface.h	包含文件
tstat_lcd.cpp	LCD 功能
tstat_lcd.h	包含文件
tstat_util.cpp	实用功能
tstat_util.h	包含文件

12.6.2 软件描述

根据功能不同，我们可以将调温器软件划分为用户界面、控制逻辑、显示管理三部分。其中，用户界面部分提供读取旋转编码器与在不同显示屏之间进行导航的功能，对应的源代码模块是 tstat_iface.cpp。tstat_lcd.cpp 中包含处理 LCD 显示的功能。源代码模块 tstat_ctrl.

cpp 提供借助 DHT22 传感器感知温度与湿度的功能、控制器逻辑，以及用户定义程序。

本项目中用到了许多库，它们分别用于支持 DS1302 RTC、旋转编码器、时间日期功能、AVR 定时器 1 的外围设备，如表 12-6 所示。使用 Arduino 的好处之一是，如果需要某个模块、传感器或扩展板的支持库，那么很可能有人已经为它写好了库，所以不必从零开始编写，直接使用现成的库即可。

表12-6：调温器软件中用到的外部库

名称	功能	作者
Time	时间功能	Michael Margolis
DS1302RTC	RTC 类	Timur Maksimov
ClickEncoder	旋转编码器类	Peter Dannegger
TimerOne	定时器 1 类	Lex Talionis

你可以在 Arduino Playground（http://playground.arduino.cc）与 GitHub（http://www.github.com）上找到这些库。下载之后，请认真阅读相关说明文档及源代码，进一步加深对代码功能与配置的理解。DHT22 库与第 10 章制作 GreenShield 时使用的那个库是一样的。

tstat.h 文件包含全局定义，其他模块会用到它们。这些定义涵盖图 12-7 中的引脚分配，如示例 12-1 所示。

示例 12-1　tstat.h I/O 引脚定义

```
#define ROTENC_A    0
#define ROTENC_B    1
#define ROTENC_SW   2

#define LCD_D4      4      // 由LCD扩展板预定义
#define LCD_D5      5      //
#define LCD_D6      6      //
#define LCD_D7      7      //
#define LCD_RS      8      //
#define LCD_E       9      //
#define LCD_LED     10     //
#define LCD_SW      A0     //

#define RTC_SCLK    11
#define RTC_IO      12
#define RTC_CE      13

#define RY1         15      // A1
#define RY2         16      // A2
#define RY3         17      // A3
#define RY4         18      // A4

#define DHT1        3       // 内部DHT22
#define DHT2        19      // A5,外部DHT22
```

示例 12-2 显示的是 Thermostat.ino 中的 `setup()` 函数，它用于初始化 LCD，显示启动信息，检查 RTC 模块，以及显示第一个屏幕。

示例 12-2　setup() 函数

```
void setup()
{
    lcd->begin(16, 2); // 设置LCD大小

    TitleDisp("Initializing...", "", 1000);

    lcd->clear();

    if (rtc->haltRTC())
        lcd->print("Clock stopped!");
    else
        lcd->print("Clock working.");

    lcd->setCursor(0,1);
    if (rtc->writeEN())
        lcd->print("Write allowed.");
    else
        lcd->print("Write protected.");

    delay (2000);

    // 设置时间库
    lcd->clear();
    lcd->print("RTC Sync");
    setSyncProvider(rtc->get); // 从RTC获取时间
    lcd->setCursor(0,1);
    if (timeStatus() == timeSet)
        lcd->print(" Ok!");
    else
        lcd->print(" FAIL!");

    delay (1000);

    TitleDisp("Initialization", "complete", 1000);

    curr_screen = 0;
    Screen1();
    disptime = millis();
}
```

Thermostat.ino 文件中的 `loop()` 函数也不复杂，如示例 12-3 所示。大部分工作发生在旋转编码器使用时，以及软件获取温度与湿度数据并执行 tstat_ctrl.cpp 中的控制功能的过程。

示例 12-3　调温器主循环函数

```
void loop()
{
    // 从RTC获取当前时间与日期
    RTCUpdate();

    if (input_active) {
        HandleInput();
    }
```

```
        else {
            // 在屏幕1与屏幕2之间切换
            if ((millis() - disptime) > MAX_DISP_TIME) {
                if (curr_screen) {
                    Screen1();
                    curr_screen = 0;
                    disptime = millis();
                }
                else {
                    Screen2();
                    curr_screen = 1;
                    disptime = millis();
                 }
            }
        }

        GetTemps();
        SystemControl();
    }
```

未使用编码器时，显示器将在屏幕 1 与屏幕 2 之间切换，依次显示当前条件、操作状态与日期时间。使用编码器时，`input_active` 标记设为 `true`，并且一直保持 `true` 状态，直到用户按下编码器按钮开关返回屏幕 1。请注意，输入屏幕处于使用状态时，系统控制仍然起作用。

12.6.3　测试

测试过程分为 3 个步骤，同时适用于原型与最终成品。第一步是设置 RTC 中的日期与时间，使用编码器按钮切换到日期与时间设置界面即可实现。返回主屏幕后，应该能看到正确的时间与日期。下一步是设置目标温度、滞环宽度（对应于显示器上的 C 参数）与工作模式。首先，验证能否手动打开系统风扇，然后输入目标温度值，并指定工作模式。周围温度达到目标温度加或减循环范围（滞环宽度）的一半时，可以在屏幕上看到相关显示，并检查系统是否停止加热或冷却。

最后，应该测试每个情景设置，确保它们都能很好地贴合你的实际应用场景。请注意，当一个情景的结束时间与另一个情景的起始时间发生冲突时，将优先考虑前一个情景的结束时间，而非后一个情景的起始时间。

接下来的两个星期里，你应该持续关注调温器的运行情况。期间可能需要对时间、目标温度做一些微调，以使设备能够更高效地工作。如果已经制作了一个 GreenShield 板，那么可以使用它记录温度、湿度，采集调温器的性能数据。还可以写一段简单的 Python 脚本，可以定期查询 GreenShield 以获取数据。

12.7　最终版本

最终版本的调温器在物理层面上与原型有一些不同，但在功能与信号连接方面都是一样的。主要不同有 3 点：最终版本使用了 Nano 开发板、裸 LCD 模块（代替原型中的 LCD 扩展板）、一个外壳（安装调温器的所有组成元件）。最终版调温器的元件列表如表 12-7 所示。

表12-7：最终版调温器元件列表

数量	元件
1	Arduino Nano（或类似设备）
1	螺丝端子原型板
1	16 × 2 LCD 显示器模块
1	四芯继电器模块
1	DHT22 温度 / 湿度传感器
1	KEYES KY-040 旋转编码器模块
1	实时时钟模块
1	塑料外壳

12.7.1 组装

最终组装主要是指，将各个元件安装在外壳上。在此之前，需要先对外壳做一些改动。先磨平顶盖上的凸印字母，去掉侧面安装突耳，再在安装显示器与旋转编码器的位置上打好孔洞。在外壳底部也要钻一些孔洞，用于安装螺丝，以及让原有的加热器与 A/C 控制线通过。经过磨平、钻孔、切割处理之后，再为外壳刷上中性乳白色。

我使用一个螺丝端子原型板安装 Nano 开发板，并将 Nano 引脚连接到相应的端子与排针上。图 12-24 显示的是 Nano 原型板的顶视图，其中包含 LCD 扩展板、旋转编码器、RTC 模块。RTC 模块将安装到外壳里面。

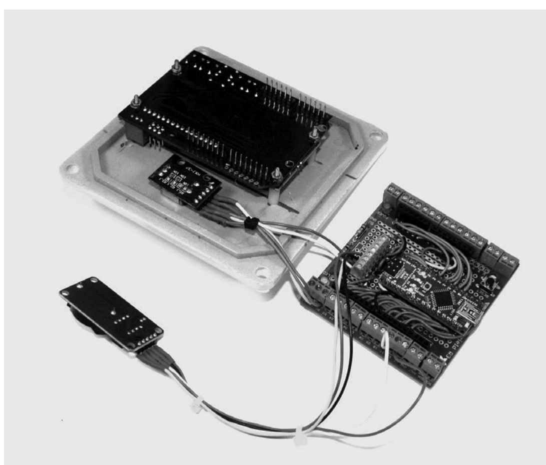


图 12-24：Nano 原型板与其他顶盖元件

由于我将所有连线都放在了原型板的顶部，所以板子底部只有焊盘。我选用最细的绝缘绞线（28-gauge 7-strand），没有使用金属绕丝线。带有 Kynar 绝缘的 Solid 30 AWG 导线很适合绕线结构，但对本项目可能并非如此。

使用螺丝端子原型板最大的好处是，可以很容易地将其他元件的连线连接到接线端子上。就调温器而言，这些元件包括 DHT22 传感器、实时时钟模块、继电器模块、旋转编码器。

图 12-25 是外壳经过一系列修改后的最终呈现。LCD 扩展板与旋转编码器已经安装完毕。图中看不到 DHT22 传感器，它安装在外壳底部外面。此外，你可能也注意到，还没有安装外盖螺丝。

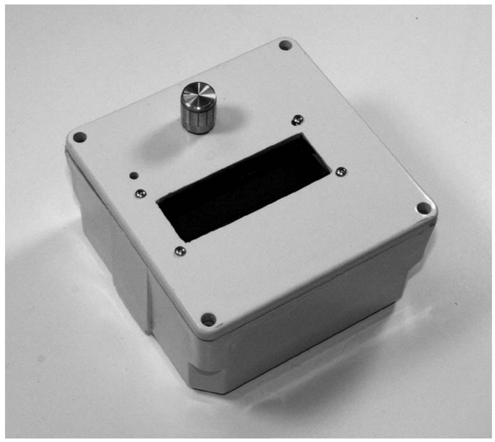


图 12-25：几乎完成的调温器外壳

LCD 显示器窗口的左上角有一个小孔，通过它可以访问 LCD 扩展板上的微调电位器，以设置显示器的对比度。在最后一步，我们将对 LCD 显示器窗口边缘做一些修剪。当然，使用透明塑料盖也是一个不错的主意。它看上去的确像工业设备上的一个部件，但正如我之前所说，外观与能否正常工作无关。我使用了一个金色旋钮（在彩色图片中可以看出），因为当时我手上可用的旋钮只有它，并且它不带有指针。

四芯继电器模块安装在外壳底部。RTC 模块安装在外壳内部。图 11-25 显示外壳中各个元件是如何安排的。现有的 HVAC 连线经由外壳底部的小孔进入内部。请注意，小孔从中心垂直偏移，以便为继电器模块留出空间。

虽然外壳显得有点小，但的确能装下所有元件。最后合上外壳之前，还有如下几件事要做：

(1) 在墙面上确定螺丝安装位置并钻孔；
(2) 将现有的 HVAC 控制线穿过底部孔，并将外壳安装到墙上；
(3) 将 HVAC 线连接到继电器模块；
(4) 连接电源并检查显示器；
(5) 扣上封盖。

如果软件已经上传到 Nano，那么调温器现在应该开始工作。调温器上没有单独的电源开关，一接上电源线就开始工作。通电之后，显示器的背光会点亮 15 s。超过 15 s 之后，背光熄灭，此时可以转动编码器旋钮再次开启背光。

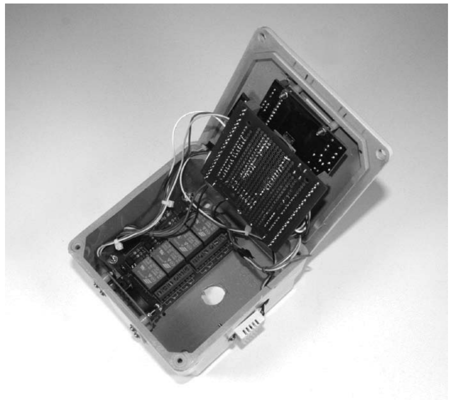


图 12-26：调温器内部元件

外壳本身带有一个橡皮垫片，如果觉得有必要，你可以使用它。我用它只是因为它可以紧密地贴合封盖，并且可以为内部元件提供更多空间，并不是为了防雨。我一点也不担心会有雨水进入调温器。

12.7.2 测试与运行

该阶段的测试包括使用编码器在显示屏之间切换，以及验证编码器的长按与短按动作能否正常工作。由于软件只加载到闪存而并不加载到 EEPROM，所以测试时可能需要重复设置参数。如果一切工作正常，你就可以开始为自己特定的使用环境设置相应的运行参数。

开始时，我建议尽量简单一些，比如只控制加入或冷却系统。当你对系统的运行状态满意后，可以进一步尝试启用白天 / 夜间情景模式，观察在这些情景模式下系统是否可以正常工作。随着对这个系统越来越熟悉，获取更多使用经验之后，你可以继续加入湿度与循环时间设置。这样的设备需要做一些调优工作，不断调整各种设置，最终让它能够在你的实际环境中高效地工作。

第二个 DHT22

图 12-4 中，除了内部 DHT22 之外，还可以看到有另外一个 DHT22 传感器输入。在调温器的第一次迭代中，我将其忽略掉，这样就可以把主要精力放在基本功能的构建上。从电路上看，向系统加入另一个 DHT22 输入很容易，但这样会进一步增加软件的复杂度。

加入第二个传感器之后，调温器不仅可以监视室内环境，还可以感知室外环境，获取室内外温差、任意给定时刻的变化速度，以及估算透过墙壁窗户散失或获取的热量。通过足够的努力与精巧的编程，调温器可以平衡各种因素，使室内温度达到理想的水平。这听上去很棒，但实现起来并不简单。

本书中的所有软件——包括调温器的源代码——都可以从 GitHub（https://www.github.com/ardnut）上下载。我会不定时地为调温器及其他示例项目的软件推送更新。

12.8 成本明细

本项目的目标之一是，尽可能多使用现成的开发板与模块。这不仅可以节省大量时间与精力，还有助于减少支出（定制 PCB 可不便宜）。从制作过程看，制作本项目耗费的精力要比第 11 章的信号发生器少得多。表 12-8 列出了制作调温器时需要的主要元件。

表12-8：调温器各元件价格列表

数量	元件	来源	单价	总价
1	Arduino Nano	SainSmart	$14.00	$14.00
1	螺丝端子原型板	Adafruit	$15.00	$15.00
1	RTC 模块	DealeXtreme	$2.00	$2.00
1	旋转编码器模块	Various	$6.00	$6.00
1	LCD 显示器扩展板	SainSmart	$10.00	$10.00
2	DHT22 传感器	Adafruit	$10.00	$20.00
1	四芯继电器模块	SainSmart	$7.00	$7.00
1	电气设备外壳	Home Depot	$7.00	$7.00

* 总价等于元件单价与数量的乘积。

总费用 = 81.00 美元

请注意，KEYES 模块（比如 KY-040 旋转编码器）通常都是套装模块。当然，可以从 eBay 与 Amazon 的供应商那里单独购买这些模块，但也可以购买相关套装，价格会更优惠。

正如本书中的其他示例项目，通过货比三家、寻找便宜货，你可以比表 12-8 列出的价格花费更少。我曾经见过一块 Nano 兼容板只需要 5 美元，一个 DHT22 传感器只要 7~8 美元。

12.9 后续步骤

本项目是个受限制的例子，当你试图使用现成的开发板与模块制作复杂设备时，可能会碰到这种情况。如果我打算制作第二个版本的调温器，那么会专门为它定制一块 PCB，其上包含所有元件。类似于第 10 章中的 GreenShield 板，定制的 PCB 将带有板载继电器，当然也会集成供电电源。

这种设计也是 I/O 多路复用器的主要候选对象，比如第 10 章的 Switchinator PCB 中使用的 MPC23017。可以通过 MPC23017 控制 LCD 显示器。Adafruit 公司正在销售一款带有 MPC23S17 的显示器扩展板，它是 SPI 版本的 IC。

另外，选择元件时，我会选用表面贴片元件，但继电器、DHT22 和接线端子除外。我认为继电器应该更小一些，甚至使用表面贴片式，因为通过这些继电器触点的电流其实并不大。精心设计并选择合适的元件，可以使整个 PCB 明显减小。

一些不错的特性不适合用在当前设计中，包括蓝牙或 Wi-Fi 接口、microSD 闪存卡槽（长期记录数据）、前面板上额外的 LED 灯。从目前情况看，Arduino 调温器应该算是一个物联网设备。但要真正用在物联网中，还需要对设计做一些修改与完善。在一切皆被吹捧为

IoT 设备的环境下，调温器是名副其实的物联网设备。但我不认为让咖啡壶或微波炉与电冰箱通信是个好想法，也不觉得这样做会有什么意义。

12.10　资源

表 12-9 列出了一些电子元件分销商与供应商，我从他们那里购买了制作调温器需要的所有元件。对于本书中的所有示例项目，你总有多种渠道可以选择。表中列出的这些只是我购买电子元件时经常使用的，仅供参考。

表12-9：元件供应商

分销商/供应商	URL
Adafruit	www.adafruit.com
DealeXtreme (DX)	www.dx.com
DFRobot	www.dfrobot.com
SainSmart	www.sainsmart.com

我在本地“家得宝”商店的电子器件区买到了本项目使用的外壳，你可以从任何一家电子元件商店或货源充足的五金店购买到类似的外壳。

第 13 章

模型火箭发射器：设计研究

本章内容与前面 3 章不同，之前 3 章通过示例项目向各位介绍如何将概念、工具、元件应用于实际，而本章只阐释如何进行设计研究。我们不是真的要制作什么，而是阐明一些可行的方法，确定合适的工具与组件，并评估设计的优劣。这种工程活动（engineering activity）不仅可以应用于本章的模型火箭发射器，还可以应用到其他任何复杂度的 Arduino 设计项目中。

> 请记住，本书讲解的重点是 Arduino 硬件及其相关模块、传感器、组件，书中给出的示例代码都是软件中最关键的部分，它们并不是完整可运行的代码。关于示例与项目的完整软件代码，请前往 GitHub（https://www.github.com/ardnut）进行下载。

13.1 概览

Arduino 火箭发射器是一个非常有趣的应用，原因如下：首先，它具有很强的扩展性，在 Arduino 硬件允许的范围内，你可以根据需要随意扩展（当然也要考虑费用）；其次，从物理结构看，发射器形态多样，既可以是带有开关与 LED 灯的小塑料盒子，也可以是带有数显与一两个钥匙开关的控制台；最后，除了将电流引向点火器这一简单功能之外，发射器还拥有更多功能。它可以支持 Wi-Fi 或蓝牙连接，具备对点火器的连贯性、倒计时器与发射状态汇报的合成语音生成、安全输入范围进行连续检查的能力，甚至可以在发射之前切断火箭的 DC 电源，以使内部电池维持满电状态。在发射器上要投入多少精力取决于你的目标、技能与经济能力。

在本章最后部分，你应该能够对购买什么样的硬件、软件编写与测试的难度，以及预估一个给定的设计需要付出多少努力有清晰的认识。通过前面几章的学习，相信你已经学会制作元件清单，并对设计费用与制作难度有了大致的了解。同时也学会为你的设计绘制初步框图与软件流程图。一旦掌握了这些内容，剩下的工作就是搜集所需的元件并将其组装起来，然后编写与上传软件，让你的设备开始运行。

制作火箭发射器这类设备之前，先要了解相关的安全规定。美国火箭技术协会（NAR）和的黎波里火箭技术协会（TRA）发布了安全处置与发射模型火箭的指导手册。此外，美国还有许多州出台了一些规章制度，指出发射模型火箭的时间、地点，以及哪些才算作模型火箭。

13.2 设计周期

我们设计一个中等复杂度的东西时，常常会发现自己又回到了修改或寻找替代方案的先前步骤。有时这可能意味着你要抛弃原来的设计，从头开始。其实这种情形并不少见，设计本身就是一个不断迭代的过程。随着迭代的不断进行，原先不明显的地方变得越来越清晰、明朗。我们几乎不可能预见一切。

整个设计流程遵循第 10 章给出的设计步骤。先从设计目标开始。设计目标负责创建功能需求。根据功能需求，我们可以确定要使用什么样的硬件与软件才能实现目标需求。然后了解所选元件与技术的细节，评估它们是否真的合适。图 13-1 显示了设计周期包含的内容。

每一步结束时，我们都会停下来，思考之前都做了些什么。如果有问题，可以立即想想该如何解决。根据问题的类型与严重程度，我们可以退回上一步进行修改，或者放弃一些非必需的部分，降低预期目标，继续前行。在某个特定步骤的活动中，也可能发现一些问题，比如买不到某个特定的扩展板、传感器或模块，并且无法自己制作。这种情况需要停下来，回想起初为什么那样选，并想想是否可以根据当前情况做相应修改。

有时可能并不需要真的制作原型。设计主要由现成的模块组成时，经常会出现这种情况，许多基于 Arduino 的设计也是如此。我偏爱使用原型平台（自己制作或购买）制作原型，比如第 10 章中使用的 Duinokit 与免焊面包板、第 11 章与第 12 章中使用的板载 Arduino 板。使用这些原型平台时，可以根据需要灵活连接任意元件，不必焊接导线与打孔，并且可以得到与最终成品一样的功能。也许你认为可以在某些设计项目中不制作原型，但请注意，一旦最终产品出现问题，你将不得不耗费更多时间金钱进行解决。

在现实情形下，处理设计或实现中的缺陷（硬件或软件）花的费用会随着项目进展的步骤（如图 13-1 所示）数的增加而增长。换言之，修改无效的功能需求要比重写带有缺陷的软件或修改硬件设计缺陷便宜得多。同样地，在原型中处理问题要比在即将投入生产的设计中处理问题更容易、更便宜。这就是制作原型如此受欢迎的原因所在。

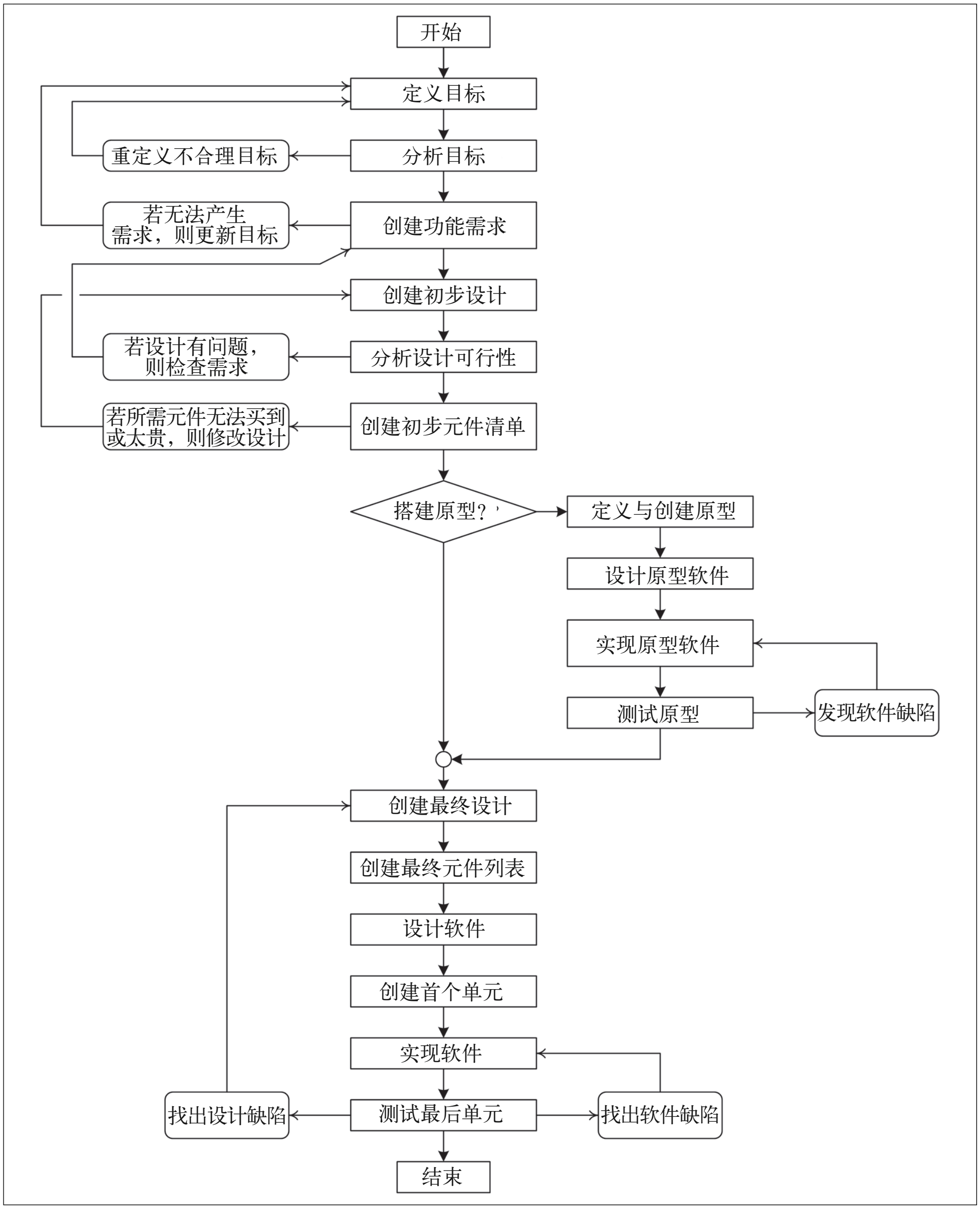


图 13-1：设计周期

13.3　目标

设计活动的第一步是，确定设计要实现的目标。刚开始时，我们可能尚未掌握所有细节，尽管如此，依然可以决定一些最基本的特征。虽然只是很小的一部分，但随着时间的推

移，目标列表将变得越来越长，有时这些变化很富戏剧性。事实上，让设计目标符合实际通常并不容易，特别是制定目标的人与最终设计的人不是同一个人时，难度会更大。幸运的是，这里并没有出现这种情况。只要我们保持热情、乐观，不脱离实际，一切都会很顺利。

制作目标清单时，我建议你根据目标的重要程度进行组织。换言之，你要将“必须有”（must have）的功能放到列表最上面，而把“可有可无”（bells and whistles）的功能放到列表最下面。这样会帮助你从列表中筛选掉一些不必要的功能，从而保证费用支出、时间与精力不超过允许的范围，同时又不会失去必备功能。

下面是我制作火箭发射控制器（亦称火箭发射器）时列出的要优先实现的目标，它们是本项目的主要目标。

(1) 发射控制器包含倒计时器，并提供时间预设功能（5 s、10 s、15 s、30 s 与 60 s）。
(2) 出于某些原因（后面给出），你可以在任意时刻终止倒计时。
(3) 发射控制器必须配有钥匙操作的安全联锁开关。只有联锁装置处于 on 位时，发射器才启动计数或激活点火器。
(4) 按下按键开关，发射倒计时器才启动。
(5) 整个倒计时期间，按倒计时启动按钮之前，必须先按下安全开关，发射倒计时才启动。可以是手持式附件或小盒子，必须与倒计时启动开关分离。
(6) 若安全开关在发射之前释放，则终止倒计时，控制器进入安全模式（点火器电路被禁用并接地）。
(7) 发射器必须有能力同时激发多个点火器，最少 2 个，最多 8 个（若可能）。
(8) 对每个点火器的监控应该独立于其他任何点火器，以便在发射之前检测电路连续性问题。
(9) 任何一个点火器的连续性故障都将终止发射（过早释放安全开关会发生同样的反应）。
(10) 控制器使用闪烁的 LED 灯跟踪倒计时（读秒阶段）。其他 LED 指示控制器的状态（等待发射、倒计时、安全中止、点火器故障）。

次要目标罗列如下。

(1) 七段 LED 显示器显示倒计时
(2) 七段 LED 显示器显示当前时间
(3) 16 × 2 LCD 显示器显示系统状态数据与错误信息
(4) 所有元件安装于一个矮型斜顶控制机箱中。

下面这些目标是可选的。

(1) 发射控制器有能力在倒计时序列的某个点上激活外部设备，比如断开电源、拉响警笛、启动板载设备（像开启摄像机等）。
(2) 控制器带有一个连接器，用于连接一个较大的七段显示器，显示组事件。
(3) 控制器有能力为倒计时与状态公告生成合成语音。
(4) 蓝牙接口用于连接到远程天气监测系统（主要获取发射区的风速）。如果发射器具备该功能，那么它也必须有能力在风速超过警戒值时终止发射，并将这一情况报告给发射总指挥（用户）。

上面目标列表包含的内容相当多，并且“野心勃勃”。其中，实现主要目标看上去并不怎么难，这些目标功能是我们一定要实现的；而次要目标的实现则有一定难度；可选目标让人觉得很有趣，但实现起来需要创建者付出更多精力与耐心（更不要说会花费更多金钱）。

13.4　选择与定义功能需求

我罗列目标的方式使得我们可以对其稍作修改即可直接生成功能需求。功能需求必须清晰明确、简明扼要、条理分明，并且是可验证的。有时，目标措辞和功能需求差不多，所以可以很容易地定义功能需求。

现在最大的问题是，要将多少目标确定为功能需求。为了回答这一问题，需要使用某种评分系统判断达成某个目标的难度、费用以及必要程度。

下面从主要目标开始。表 13-1 列出了我对各个主要目标的评估，分别从实现难度、费用、必要性三个方面进行。每个方面分值为 1~5，1 是最低分，5 是最高分。

表13-1：主要目标排行

目标	难度	费用	必要性
1.1	2	2	2
1.2	3	2	5
1.3	3	2	5
1.4	2	2	4
1.5	2	3	5
1.6	2	1	5
1.7	3	3	3
1.8	4	3	5
1.9	3	2	5
1.10	3	2	2

由表 13-1 可以得出这样的结论：目标 1.1（可预置的倒计时器）可以被替换为固定的 10 s 或 15 s 计数（如果有必要）。另外，很显然，目标 1.10（LED 闪灯）可以被归入“可有可无”的类别或者次要目标，若有必要，也可以将其去掉（尽管如此，我们仍然希望至少保留一个倒计时 LED 灯，以告知当前正在发生什么）。

同样对次要目标与可选目标进行评分，如表 13-2 所示。

表13-2：次要目标与可选目标排行

目标	难度	费用	必要性
2.1	3	3	2
2.2	3	3	2
2.3	3	3	2
2.4	3	3	2
3.1	4	4	1
3.2	4	3	1
3.3	4	4	1
3.4	4	4	1

你可能已经注意到，表 13-1 与表 13-2 中出现一个有趣的现象——主要目标实现起来相对容易，并且花费不高。另外，除了 1.1 与 1.10 之外，其他主要目标都是必需的。表 13-2 中，实现各个目标的花费与难度增加，而必要性则降低。其实，可选目标都不是必需的，不过实现后会很有趣。

我写下这些目标时，并没有像上面那样特意安排，只是想到什么就写什么。说实话，这是我多年来不断这样锻炼得到的结果。通常在组织项目目标时，首先依据必要程度，然后是实现的难度或成本支出。这具体取决于什么对你更重要：金钱还是时间。

在这种情况下，我们都会面临取舍的问题。比如某个目标是必需的，但实现成本很高，此时应当去掉一些不必要的目标让项目预算保持合理呢？还是多花些钱把这些目标都实现呢？有时，如何选择很简单，比如在主目标与可选目标之间做选择。经过权衡，最终制作的发射器可能没有原先想得那么酷炫，但我们也不能为此而倾家荡产，只要它能出色完成发射火箭的任务即可。从另一方面来说，对于两个同等重要、花费也一样的主要目标，比如目标 1.1 与 1.10，在两者之间选择时就要考虑各个目标的实现难度。就火箭发射器而言，安装一个旋转开关选择倒计时时间可能会更容易，而为 LED 灯打孔、连线、安装并编写软件进行控制的难度则相对要大一些。

为了继续做设计分析，我们假设保留所有项目目标，以便了解它们在后面步骤中如何推动设计，以及如何影响对元件的选择。表 13-3 是根据目标列出的功能需求。

表13-3：火箭发射器功能需求

功能需求	描述
1.1	发射控制器集成一个倒计时器。倒计时到 0 时，启动点火器
1.1.1	倒计时器有 5 个预设时间段，5 s、10 s、15 s、30 s、60 s
1.1.2	使用 5 位旋转开关选择倒计时时间段
1.1.3	倒计时中改变倒计时时间段，则终止倒计时
1.2	出于某些原因（后面给出），可以在任意时刻终止倒计时
1.2.1	如果有本该关闭的点火器电路打开，倒计时将终止。（见 1.7.1 与 1.9.1）
1.2.2	倒计时时若改变倒计时时段，则倒计时终止
1.2.3	倒计时期间，若安全开关被释放，则倒计时终止
1.3	发射控制器必须配有钥匙操作的安全联锁开关
1.3.1	只有联锁开关处于 on 位时，发射器才启动计数或激活点火器
1.3.2	倒计时期间，若联锁开关被拨到 off 位，则倒计时终止
1.4	通过按按键开关启动发射倒计时器
1.4.1	launch 按钮用于启动倒计时
1.4.2	只有发射场安全开关开启，launch 按钮才可用
1.5	发射器使用发射场安全开关控制倒计时与发射
1.5.1	只有安全开关先于发射按钮按下，发射倒计时才可以进行

（续）

功能需求	描述
1.5.2	整个倒计时期间，发射场安全开关必须手动接合
1.5.3	发射场安全开关可以是手持式附件或小盒子，必须带有至少 10 feet（3 m）电缆线
1.6	出于某种原因倒计时被中断时，发射器将进入“安全模式”（safe mode）
1.6.1	在安全模式下，所有点火器断电并接地
1.6.2	退出安全模式，需要使用安全联锁开关取消发射器的安全警戒状态
1.7	发射器必须有能力同时激发多个点火器
1.7.1	发射器最多支持 6 个有源点火器电路
1.7.2	旋转开关用于选择有源点火器电路的数目
1.8	对每个点火器的监控应该独立于其他任何点火器，以便在发射之前检测电路连续性问题
1.8.1	每个点火器电路必须有连续性，以便验证点火器正常连接
1.8.2	点火器电路中，用于确定连续性的感测电流不应该超过 250 μA
1.9	任何点火器连续故障将中断倒计时器
1.9.1	如果任何一个有源点火器电路不连续，则倒计时不启动
1.9.2	倒计时期间，若任何一个有源点火器电路失去连续性，则倒计时中断
1.9.3	发射器将使用 LED 灯指示点火器电路状态（见 1.1.2）
1.10	控制器将使用 LED 灯指示倒计时与相关状态
1.10.1	当倒计时进入读秒阶段，将通过 LED 闪烁进行提醒，每秒闪一下
1.10.2	一组 LED 灯用于显示状态，以及点火器电路的连续性
1.10.3	使用一个 LED 灯显示发射场安全开关的状态
1.10.4	使用一个 LED 灯指示安全范围中止
1.10.5	使用一个 LED 灯指示联锁开关状态
1.10.6	使用一个 LED 灯闪烁指示发射器中止并进入安全模式
2.1	使用七段 LED 显示器显示倒计时
2.1.1	使用一个 2 位的七段红色 LED 显示器显示倒计时
2.1.2	成功发射后，倒计时显示为破折号字符
2.1.3	发生终止事件时，显示为 Abort
2.2	一个七段 LED 显示器用于显示当前时间
2.2.1	使用一个 4 位七段式黄色 LED 显示器显示当前时间，格式为 HH:MM
2.2.2	通过发射器前面板上的按钮设置显示的时间
2.2.3	时间显示格式采用 24 小时制
2.3	使用一个 16×2 的 LCD 显示器显示系统状态数据与错误信息

（续）

功能需求	描述
2.3.1	一个 16×2 的 LCD 显示器显示就绪状态与错误信息
2.3.2	LCD 显示器也可以显示来自其他外部设备的信息，比如来自远程气象站的数据
2.4	所有元件安装于一个矮型斜顶控制机箱中
2.4.1	一个矮型金属机箱用于安装整个发射器系统
2.4.2	机箱至少为 14 in 宽、10 in 深（35.5 cm×25.5 cm），机箱前部高度为 1.5 in，机箱后部高度为 3 in（3.8 cm×7.6 cm）
3.1	发射控制器有能力在倒计时序列的某个点上激活外部设备
3.1.1	倒计时中，控制器可以在用户指定的时间点上通过继电器激活外部设备
3.1.2	行动时间点（action points）通过 USB 连接到发射器的 Arduino 上进行设置
3.2	控制器带有一个连接器，用来连接一个较大的七段显示器，显示组事件
3.2.1	到内部两位倒计时显示器的信号会被复制到机箱后面的连接器
3.2.2	外部显示器信号由大电流 IC 控制，以提供足够的驱动电流
3.3	控制器有能力为倒计时与状态公告生成合成语音
3.3.1	发射器集成了一个语音合成扩展板，用于产生易懂的语音
3.3.2	语音输出由用户定义
3.4	蓝牙接口连接远程天气监测系统
3.4.1	用户可以定义风速上限，超过风速限制将终止发射
3.4.2	前面板的一个 LED 灯用于指示风速超限终止发射

编写上面这些功能需求时，我尽量把它们与前面的项目目标逐条对应，但仍有一些偏差，主要是因为我删除了一些多余的部分。

本项目设计用到了“发射场安全开关”（range safety switch），在倒计时期间，它必须被按下（激活）。这种安全措施是必不可少的。发射现场通常有两个人员，一个是发射主管，一个是发射场安全员。其中，发射场安全员的主要工作是观察是否有影响发射的因素（比如发射场的风速超过限制、有无关人员闯入等）出现，若有则立即终止发射。

13.5 进行初步设计

前面已经定义了功能需求，通读这些需求后，会从中发现一些明显的设计准则。本项目中，我们将需要两个旋转开关、一个 LCD 显示器、两个七段 LED 显示器、一个实时时钟、带有按键开关的小盒子（发射场安全开关）、一些背面板连接器、一些继电器，以及各种颜色的 LED 灯。

为了进一步了解设计中都包含些什么，我们可以创建一个初始框图，如图 13-2 所示。从框图中可以很明显地看到 Arduino Uno 不能处理所有 I/O 请求，我们应该考虑使用

Mega2560。此外，可能还要考虑使用多个 Arduino 来分担这些工作。

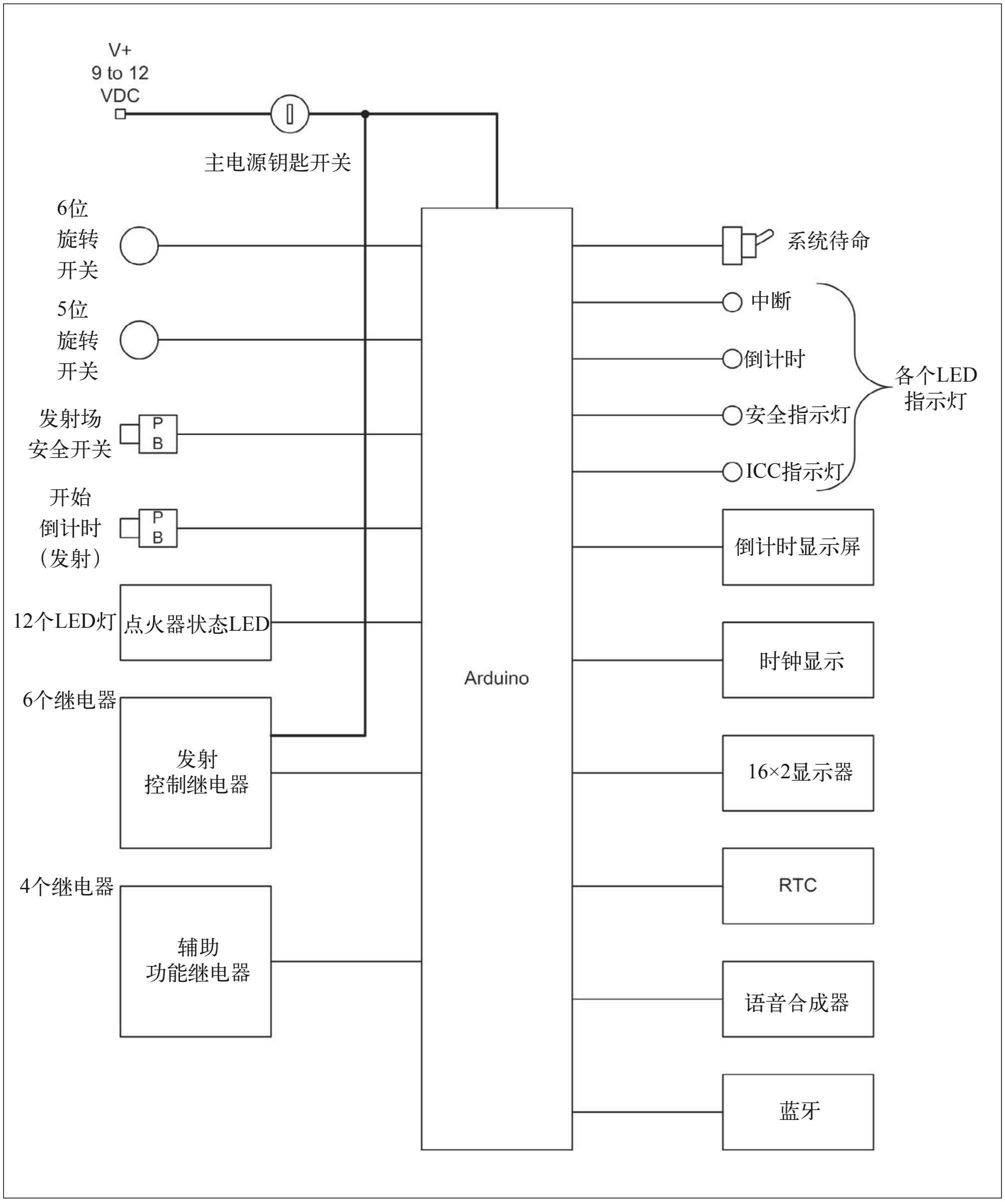


图 13-2：初步发射器框图

图 13-3 是改进后的框图。可以看到，一个 Arduino Uno 进行语音合成，另一个 Arduino Uno 处理蓝牙，一个 Mega2560 用作主控。Mega2560 中使用的 ATmega2560 MCU 拥有多个 USART 接口，这样每个板外 Uno 都可以拥有自己专用的通信通道。

图 13-3 改进了几个连接外部模块的接口，并且指定为 I2C 类型。另外，数字 I/O 端口（DIO）被特意区分出来。还使用了一个 I/O 扩展板（类似于第 10 章 Switchinator 中的那个）以控制继电器与 12 个点火器状态 LED 灯。

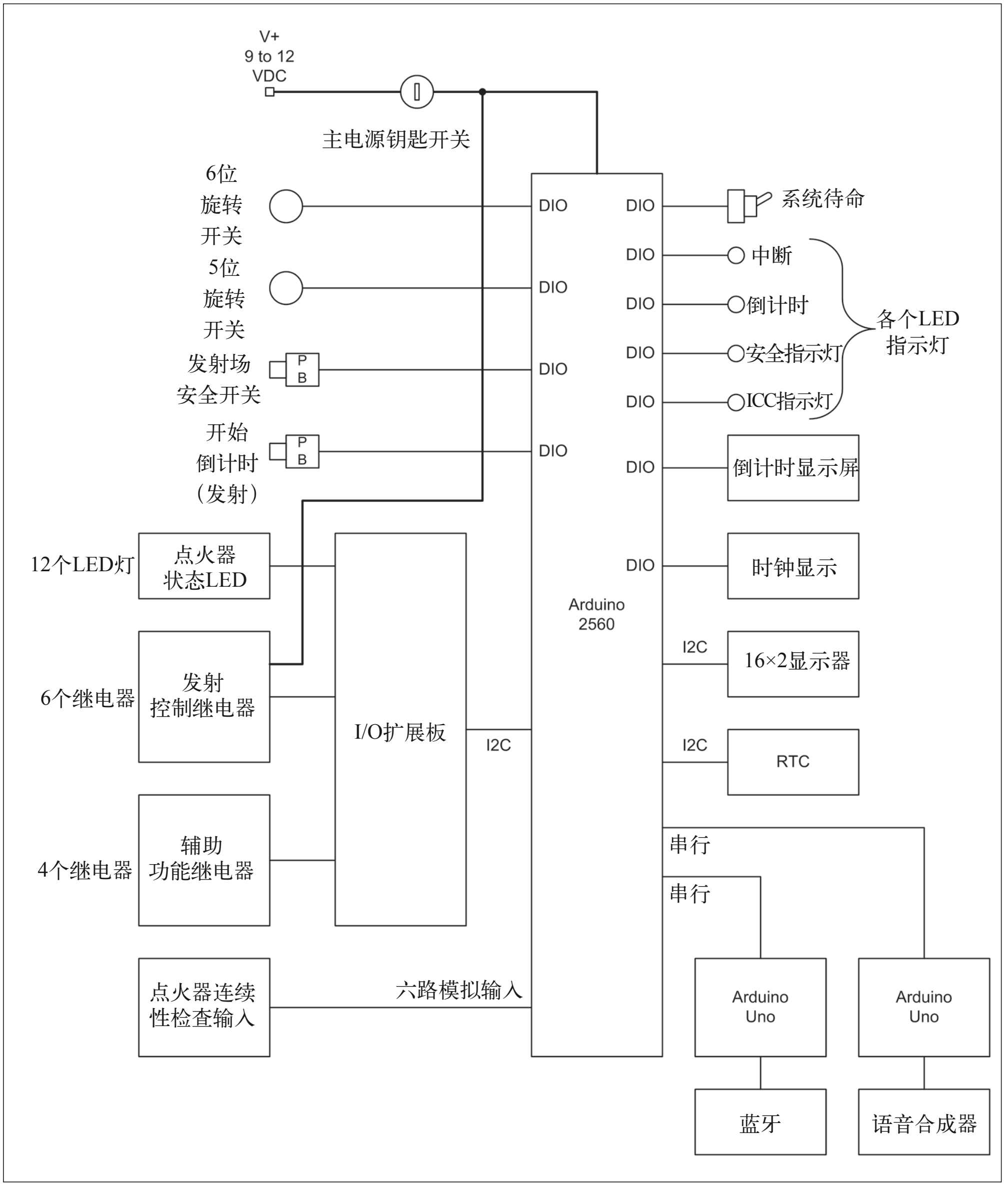


图 13-3：改进后的发射器框图

动手制作还是购买？

做一个中等复杂度的项目时，你可能需要做一些取舍：购买现成的硬件模块？还是自己设计并制作符合项目特定需要的模块？有时很难做出选择，因为这不只是费用的问题。

现成的商业（COTS）组件或模块可能拥有你需要的所有功能，也可能只有一部分功能才是你需要的。还有可能的是，这些现成组件或模块拥有你需要的功能，但连接器不对，或者情形正好相反。也有可能大小根本不合适，或者在外形上只相差一点点。又或许这些现成模块需要的电源与设计中的其他元件不同，为此需要使用其他电路进行适配。

那么，应当调整我们的设计，以使用现成模块（COTS）更有意义？还是根据实际需要自己制作更好呢？如果选用现成模块，只需做很小的一点改动。那么更明智的做法是，稍微改动后直接使用现成的模块，而不是花费时间与金钱自己动手制作。不过，如果现成模块与设计需求相差太大，比如外形差别太大，或者现成模块中带有不需要的功能，而这些功能可能引发冲突，那么此时更有意义的做法是自己动手从零开始制作合乎实际需要的模块，这样做也有利于与项目的其他组件保持一致。

如果你仍然无法做出选择，那么可以从费用支出与时间花费两个层面进行考察，究竟是使用现成组件（COTS）划算还是自己动手制作划算，哪个划算选哪个。在关键时刻，还可以使用另外一个判断准则帮你投出决定性的一票：你需要对产品以及所谈组件的内部功能拥有多大控制能力。如果你制作的是一个商业产品，并且生产时使用的现成模块（COTS）可能停产，那么在此情形下，你可能要考虑自己制作所需的模块。类似地，如果想深入了解自己产品中各个模块的内部工作原理，那么就不应该在自己的产品中使用现成模块（COTS），因为这些现成模块对你来说是个“黑盒子”（blackbox），不可能了解它们的内部细节。为此，你可能需要自己动手制作，而不是使用现成的模块。

13.5.1 设计可行性

那么，我们提出的设计现实性如何呢？它能满足我们提出的功能需求吗？当然，这是毫无疑问的。是不是设计得有点过度了？也许吧。现在回过头去，用批判的眼光认真审视初始设计，去掉一些不必要或者无足轻重的功能。

可有可无的功能

在工程领域，人们经常使用 Bells and Whistles 这个术语描述那些可有可无的功能。它们不是系统的核心功能，而只是为最终产品添加一些“装饰”“酷炫”的功能。比如咖啡机上的彩色 TFT 显示屏，它真的有助于咖啡机做出更好喝的咖啡吗？不，当然不能。这样的咖啡机放在你的厨房里会显得更酷，更有科技感吗？是的，这正是使用彩色 TFT 显示屏的原因所在，也是你愿意花更多钱购买它的原因。

想想看：现代汽车正变成一个带有各种花哨装饰的移动“宫殿”。一辆车真正需要的不过是 4 个轮子、一台发动机、一些座椅，以及一个带有车窗的车身（防雨、防风、防尘）。没人真的需要带有 10 个扬声器的音响系统、免提电话、导航系统、多位置电动座椅——如果只是为了开车上班或者去杂货店，对吗？一辆不带有各种配件与小玩意的汽车看上去可能很丑陋，但它仍然能带你去想去或需要去的地方。更好的是，这辆平凡又丑陋的汽车可能还在跑，而那辆带有各种花哨装饰的座驾可能还呆在经销商的整修车间里等待修理。

最后的结论是，简单的东西不仅更便宜，而且往往更可靠（很少出现故障）。所以，审查设计可行性时，有一点很重要：不要只关注技术层面的可行性，还要看看哪些功能不是设备正常运行所必需的，因为它们最终可能导致设备早期故障。

可行性是一种相对的看法。换言之，提到“可行性”要指出相对于什么而言。一些东西对于只在厨桌上工作的人可能没有可行性，而对于另一个有权进入整个店铺的人可能是完全可行的。预算限制是另一个要关注的重要问题，投入水平的限制也一样。一个人如果预算少并且时间有限，那么他就不该指望自己能够与另一个可以投入大量金钱与时间的人做到完全一样的程度。我没有帮手，没有太多空闲，手头的资金也很有限，所以必须慎重考虑自己能投入多少钱，希望多长时间把它做好。我相信，遇到类似问题的人不只我一个。

对一个设计及其功能特性进行可行性评估时，我喜欢使用如下 5 个问题：

- 该特征或设计在功能上有意义吗？
- 成本花费合理吗？
- 有哪些组件会很难买到？
- 在功能或设计上有任何不安全因素吗？
- 在合理的期限内，一个人就可以进行组装、编写软件并进行测试吗？

如果有任何一个问题的答案为“不”或者“不确定”，那么应该考虑去除讨论的组件或设计特性。

火箭发射器的设计（见图 13-3）真的很棒，也很了不起，但我认为无法在合理期限内完成。更好的方法是让它简单（尽可能简单），并做一个有着良好扩展性的设计，以便日后对其再次扩展。

由于编写发射器目标时组织得当，我们可以容易地对设计进行调整。第 3 级别定义的功能需求要求有较多的时间与金钱投入，并且并不属于发射器的核心功能（发射模型火箭）。所以，我打算将第 3 级别中定义的所有需求指定为未来可选的附加功能，在以后的扩展中实现。

下面看看最基本的发射器，它由级别 1 与级别 2 的功能需求组成，我们把它称为 Phase I 版本。以后会对设计做进一步扩展，做出 Phase II 版本的发射器。我们思考 Phase I 版本的设计时，也要考虑如何进一步简化设计，但前提是不牺牲任何所需的功能。

例如，图 13-3 中有一个 I/O 扩展板，它用来做什么呢？它类似于第 8 章介绍的 Centipede I/O 扩展板，拥有 64 个数字 I/O 引线，我们可以很容易地使用其中一个控制点火器状态 LED 灯、4 个系统状态 LED 灯以及 2 个旋转开关。尽可能多使用 SPI 与 I2C，这样可以进

一步简化接线。带有 I2C 接口的 RTC、16 × 2 的 LCD、I/O 扩展板都可以买得到，你也可以买到带有 SPI 接口的七段显示器模块，用于显示时间与倒计时。Phase I 版本发射器修改后的框图如图 13-4 所示。

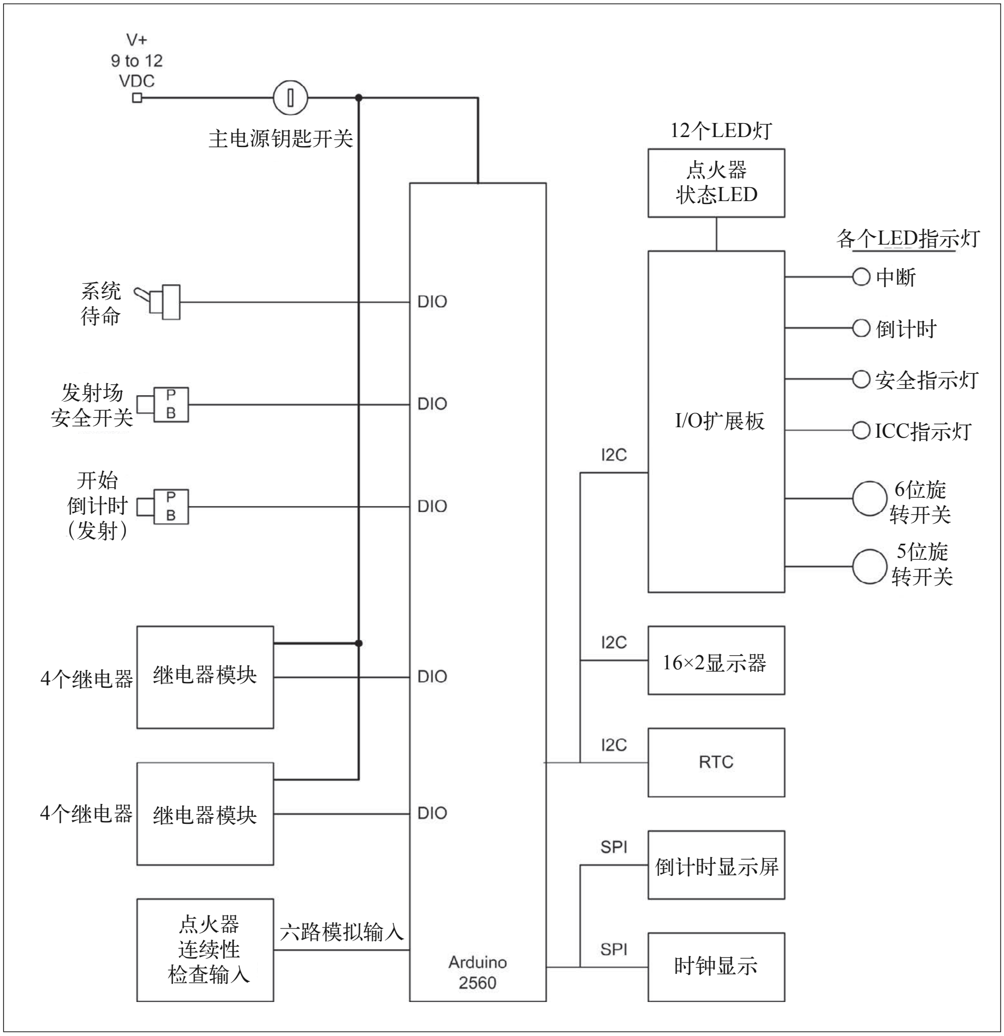


图 13-4：Phase I 发射器框图

我选择将发射控制继电器直接连接到 Mega2560 的数字 I/O 引脚上，而非通过 I/O 扩展板连接。这样，发射继电器与 Arduino 之间就没有其他控制部件了。这样做看上去并无什么好处，但的确可以从发射器的主控制线路中消除一个潜在的故障源。基于同样的原因，我们将倒计时启动装置（Count Start）、系统待命（System Arm）、发射场安全开关（Range Safety Switch）直接连接到 Arduino。此外，将开关直接连接到 Arduino 后，可以充分利用 AVR MCU 的引脚变更中断功能。当然，前提是你想这样做。

从图 13-4 中我们可以看到以后要用到什么。在将来的某个时候，尝试 Phase II 或许是可行的，但目前不行。你总会等到一个合适的时机。

13.5.2 初始元件列表

通过手中现有的初始设计（Phase I 版本），我们有足够的信息可以制作初始元件列表。通过浏览功能需求，统计各种控件与功能提及的次数，可以了解项目所需元件数量。

初始元件列表并不包含原型 PCB 等这些用于连接与安装接线端子（比如第 11 章中出现的那些）的部件。需要将其连接成一个工作单元（比如原型或最终版本的产品）时，你就会发现并记录这些需求。

表13-4：发射器初始元件列表

数量	元件	数量	元件
1	Arduino Mega2560	1	2 位七段式 LED 显示器
1	Mega 螺丝端子扩展板	1	4 位七段式 LED 显示器
1	LCD 显示器扩展板	1	钥匙开关
1	实时时钟模块	1	大按钮开关
2	四芯继电器模块	1	斜顶式金属控制台

13.6 原型

如前所述，制作原型有如下几个好处：首先，它能够让你快速验证设计正确与否，并且可以灵活、轻松地更改原型中的成员组件。如果某个模块或传感器不像预期那样工作，那么在原型中可以很容易地换掉它，而不用在金属控制台上钻更多孔进行调整。其次，对于那些涉及定制 PCB（1 个、2 个或 3 个）的设计，在寄出 PCB 设计与收到定制好的 PCB 并开始组装之间，还有一段间隔。如果你制作原型，那么可以很好地利用这段间隔期，比如编写软件或文档等。最后，有时候你手头可能已经有制作原型所需的大部分元件，将这些元件拼装起来制作最终产品的原型相对简单。图 13-5 是在松木板上使用 Arduino Mega2560 搭建的发射器的简单原型。

制作原型一定可以帮你节省时间，省去许多麻烦，但前面也提到了，有时你可以跳过这一步。就模型火箭发射控制器来说，从物理结构看，我们的设计不太复杂，没有使用定制的 PCB，所有元件都是很好理解的现成元件（COTS）。

如果你想这样做，那么完全可以轻松制作原型。如果你打算也实现可选目标，那么最好还是先制作原型，将所有元件安装到一个大板子上，确保一切能像期望的那样正常工作，然后再把它们安装到金属外壳。寻找安装板时，厨房里的砧板大小可能正合适。顺便说一下，这也是“面包板”（breadboard）这一用语的由来——此处提一下，免得你以后知道了再大惊小怪。在一个板子上安装 Arduino 开发板、接线端子以及其他各种元件有很长的历史了，仿佛让人又回到了无线电早期（20 世纪初期）。无线电发烧友们很喜欢使用面包板（它们在厨房里很常见）作为基板，安装接线柱、电子管座，以及其他电子元件。有时还会把电路图贴到这些板子上，方便安装各种元件。早期无线电大部分是用木

头建造的，无线电产品中元件的基座与面包板差别并不大，最初的面包板常用于搭建原型与测试无线电。

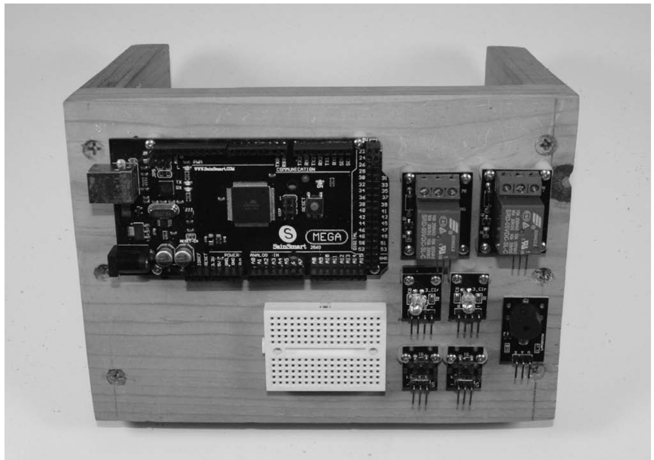


图 13-5：Mega2560 原型固定装置

虽然使用面包板搭建原型进行测试的历史源远流长，关于如何使用面包板也有很多内容值得一说，但对于本章的设计研究，我将假设我们直接到了最后一部分（关于制作原型的例子，请参考前面 3 章）。关于发射器，我们已经有功能需求（Phase I），也知道了需要用到哪些元件，接下来进入硬件与软件的最终设计阶段。

13.7 最终设计

最终设计应该体现所有功能需求，以及从原型（如果你制作了原型）获取的经验，并且可以描述最终的设备外观及其工作方式。理想情况下，最终设备与原型行为一致，软件也一样或者非常接近。

13.7.1 电气特征

图 13-6 的电路图几乎包含了发射器的所有电路，但不包含点火器连续检查电路（见图 13-7）。请注意，RTC、LCD 显示器、I/O 扩展板都使用 I2C 接口，两个七段式显示器使用 SPI 接口，只有继电器使用离散的数字输出。

模型火箭的标准点火器需要 0.5 A 左右或者更大的电流执行点火操作。图 13-7 中的电路可以判断点火器电路是打开的还是关闭的，这是通过测量 sense 端的电压实现的。这样做的好处是，可以在不必应用足够电流让点火器点火的情形下，感知点火器的打开或关闭状态。使用这个电路时，只有大约 2 mA 电流经过点火器，4.9 V 的齐纳二极管会防止发射全电压回到 Arduino 模拟输入。

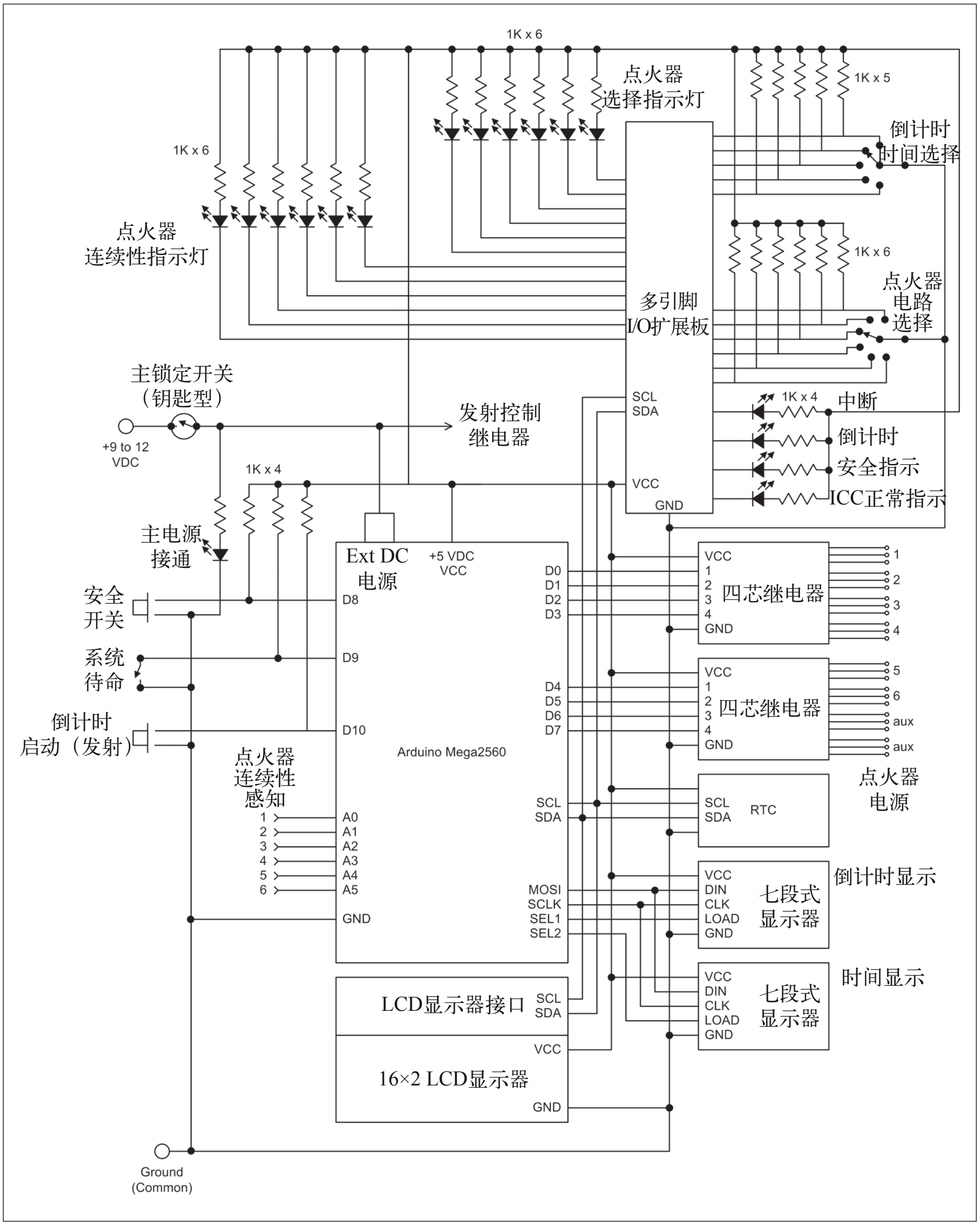


图 13-6：最终电路图（Phase I）

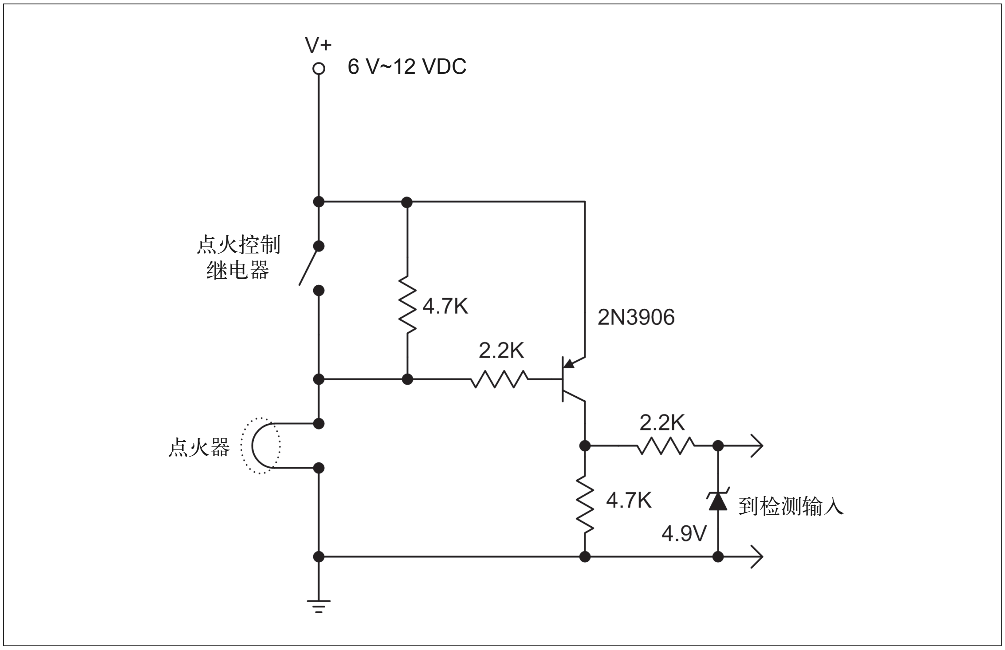


图 13-7：点火器连续性检查电路



此外还有更多点火器连续性检查电路可用，一些电路结构非常复杂（好像有个小作坊专门制作模型火箭发射控制器和连续性测试电路）。如果想了解更多相关内容，可以阅读 J.R. Brohm 关于点火器连续性测试技术的详细研究（http://bit.ly/brohm-igniter）。

图 13-6 的电路图中，有 6 个连续性电路（continuity circuits，图中标记为 Igniter Continuity Sense）连接到 Mega2560 的 6 个模拟输入。为此，我建议使用一个原型板与一些 0.1 in（2.54 mm）的接线端子制作 6 个完全一样的电路，就像我们在第 11 章中制作输入保护时所做的一样。

接线柱（类型类似于那些将扬声器连接到高档立体声接收器的接线柱）可以将点火器连接到发射控制器。同时，我也建议使用接线柱连接外置电池。如果打算在发射控制器外壳中装电池，那么需要使用另外一对接线柱将内置电池连接到点火器电路。这将允许你选择要使用的电源，具体取决于要激活的点火器的数量。图 13-8 显示了如何将点火器连续性检查电路连接到点火器接线柱。

图 13-6 中标有 Igniter Continuity Indicators 与 Igniter Select Indicators 的区域中，共有 12 个 LED 灯，它们都连接到 I/O 扩展板，用于指示有多少点火器电路激活，以及在任何给定时刻处于何种状态。这由 6 位旋转开关 S6（可设置为 1、2、3、4、5、6）确定。相应的连续性 LED 灯亮起时，表示选择的点火器接通并做好准备。发射结束后，连续性 LED 灯会熄灭，只有主动选择的 LED 灯仍然亮着。

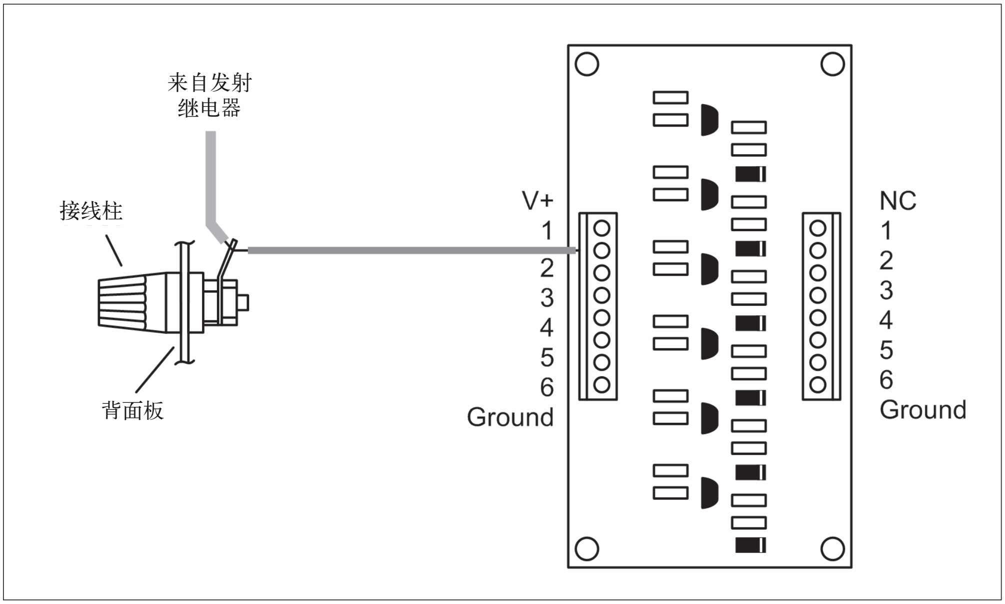


图 13-8：连接点火器检测电路板

图 13-9 从另一个视角显示了接线情况，重点强调点火器输出、LED 与开关上拉电路的穿孔板模块，以及各种模块的 DC 电源接线。请注意，I2C、SPI、DIO、AIN 功能的信号线并未在图中显示。其中，带有斜线并标有数字的总线符号指示涉及多少离散信号。此外，电路图中并未显示 DC 电源与地线的接线端子等。

图 13-9 中有趣的是，将第二继电器模块上的一个继电器用作点火安全联锁。只有接通这个继电器，点火器才能通电。同时，还要注意有 4 个穿孔板模块用于安装点火器连续性检查电路，以及各种 LED 灯与开关的上拉电阻。并且使用了 0.1 in（2.54 mm）间距的接线端子，这使得将这些组件接入系统比焊接更容易。若需要，将来某个时候也可以轻松移除并替换。

根据现有电路图，可以编制一个更详细的元件列表，如表 13-5 所示。

表13-5：发射器最终元件列表

数量	元件	数量	元件
1	Arduino Mega2560	1	大按键开关
1	Mega 螺丝端子板	8	接线柱，红色
1	LCD 显示器开发板	8	接线柱，黑色
1	实时时钟模块	1	3.5 mm 插座（单回路）
2	四芯继电器模块	1	电池盒，6 节电池
1	2 位七段式 LED 显示器	1	斜顶金属控制台
1	4 位七段式 LED 时间显示器	1	钥匙开关

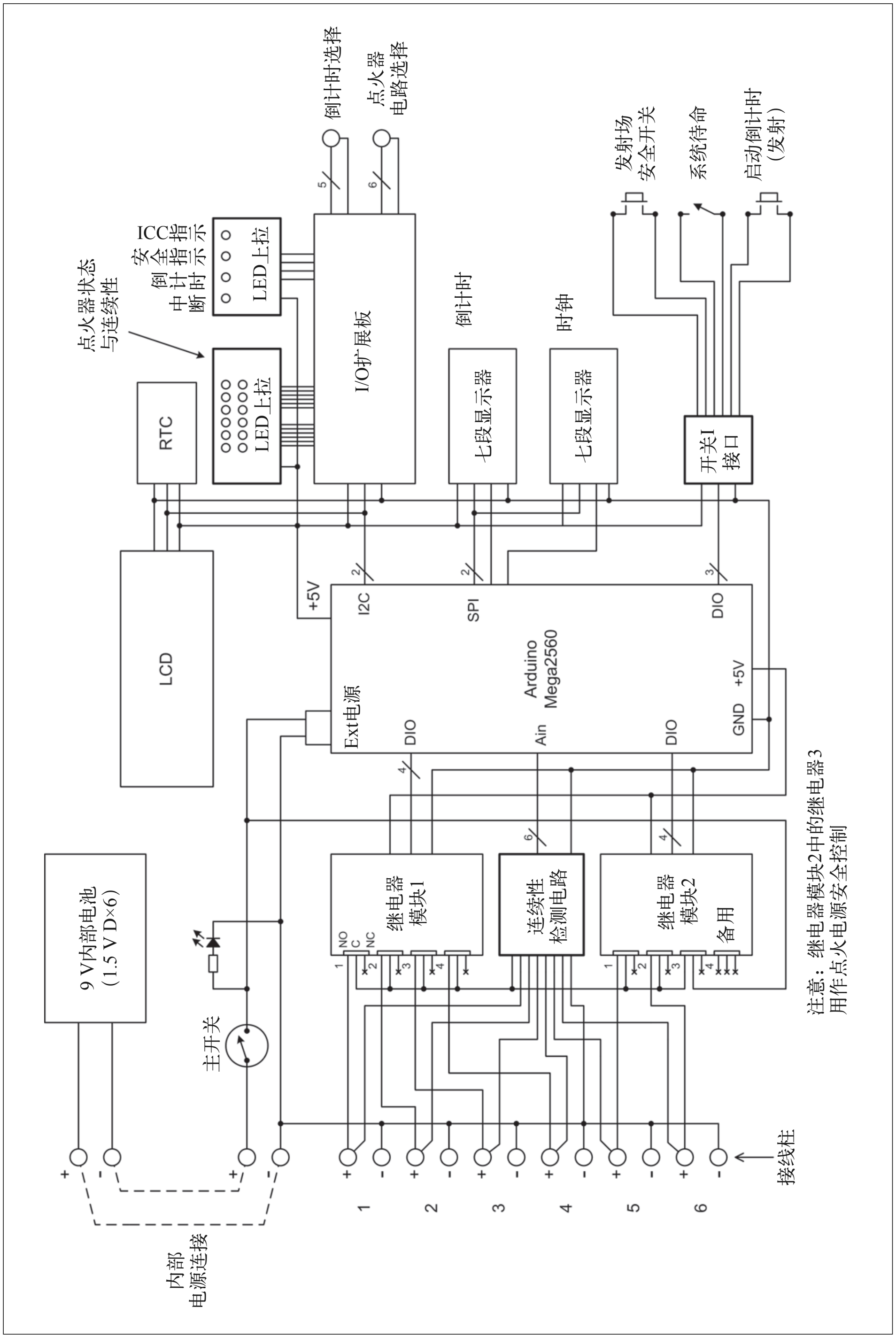

倒计时选择
点火器
电路选择
点火器状态
与连续性
ICC
安全
倒计时
中断
指示
LED上拉
RTC
I/O扩展板
倒计时
七段显示器
时钟
七段显示器
发射场
安全开关
系统待命
启动倒计时
（发射）
开关I
接口
LCD
+5V
I2C
SPI
DIO
Arduino
Mega2560
Ext电源
DIO
Ain
DIO
GND
+5V
9 V内部电池
（1.5 V D×6）
主开关
继电器
模块1
NO
C
NC
连续性
检测电路
继电器
模块2
备用
注意：继电器模块2中的继电器3
用作点火电源安全控制
接线柱
内部
电源连接

图 13-9：机箱接线图

13.7.2 物理外形

我建议将整个发射器安装在一个斜顶金属箱壳中。尽管这种形态的外壳目前不太容易买到，但图 13-10 中绘制的那种外壳还是很容易购买的。图 13-10 是一个旧机箱的示意图，它一直放在我的工作室里，没什么用。很久以前我就把里面的电子元件全部卸掉了，现在只剩一个空壳。这个机箱采用的是组合式设计（two-piece design），有 14 in（35.6 cm）宽、10 in（25.4 cm）深、前脸 1.5 in（3.8 cm）、背部 2.75 in（7 cm），顶盖后部平隔板是 3.5 in（8.9 cm）。它太丑了，我没有拍照，所以只画了个简单的示意图。

关于机箱我还想再说几句。我之所以选用这样的机箱，完全是因为我手头上恰好有。你不必非得使用，根据自己的情况选择一个合适的就好。也可以去电子元件专卖店逛逛。如果你就想用这样的，那么可以从多个经销商那里买到类似的机箱。一个新的斜脸（sloped-front）机箱价格为 30~50 美元。

选好机箱外壳之后，接下来就要设计机箱前面板的布局，确定电路板与连接器的安装位置。以图 13-10 所示的机箱为例，我建议尽量将电子元件安装在顶面板里面。这样做主要是为了避免在电路板与顶面板控件之间频繁走线。由于机箱背面只是顶面板的延续，所以需要移走顶面板露出内部元件时，不必担心会切断什么。

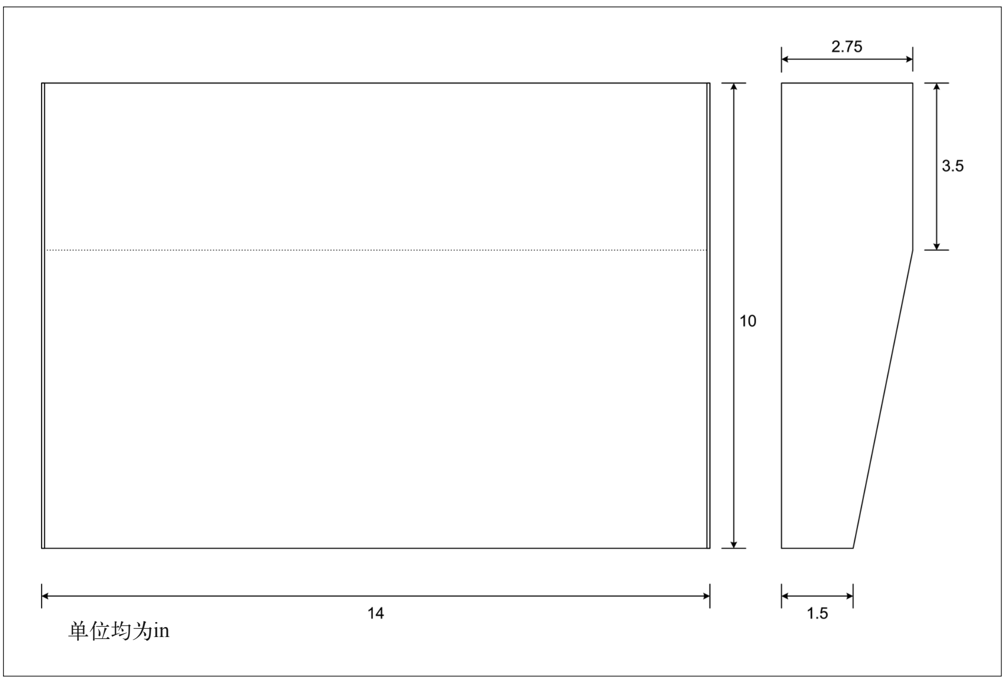


图 13-10：发射器候选机箱

图 13-11 显示了机箱面板的布局情况。请注意，这只是众多布局方式中的一种，你完全可以采用其他不同的布局方式。I/O 扩展板拥有许多 I/O 连接点（共 64 个），有很大的扩展空间。

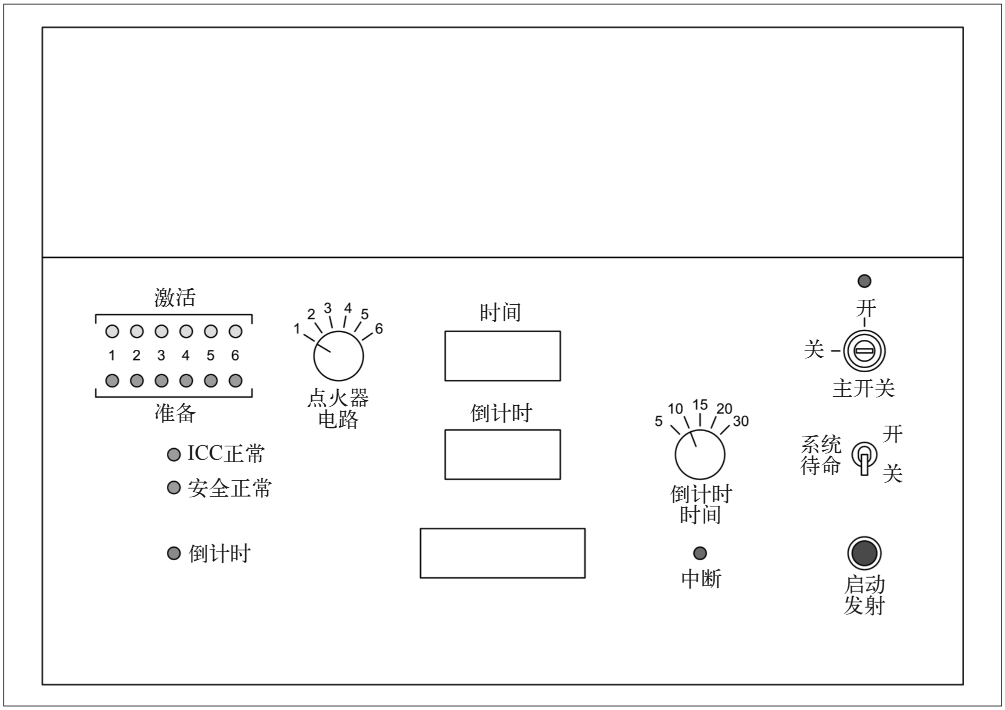


图 13-11：发射器控件面板布局示例

再说一遍，我建议将所有元件安装到顶盖里面，当然并不包括电池。这样可以很容易地组装与测试发射器，并且不必靠着机箱底部拖拉一捆导线，在顶部与底部元件之间进行连接。这样做的不利之处是，前面板上最终会出现许多螺钉头，但你可以使用油漆盖掉它们，或者干脆使用平头螺丝（前提是金属面板够厚）。

如果真的要制作发射控制器（确实有可能这么做，因为我已经为这个东西花了很多功夫），我会把 Mega2560 与 I/O 扩展板安装到机箱顶部的平隔板之下。平隔板宽度为 3.5 in（8.9 cm），内部有足够空间可以装得下一个 Arduino（或者 2 个），并且不会影响前面板上的控件与显示器。

有一种方法可以让你把电子元件准确安放到指定位置，即使用各种板子与模块的封装模型（footprint models）制作实体模型（mockup），它们由硬纸板或泡沫材料与一些不干胶标签制作而成。首先根据实际部件的大小切出形状，并使用一些圆形不干胶贴纸充当开关与 LED 灯。然后将实体模型排在一张纸上，确保面板的矩形外框排列得当。我很擅长做这个，就像有透视能力可以看穿面板一样，所以不必把模型翻过来就能知道将 LED、LCD 显示器放在哪里，把 LED 灯与开关安装在哪里。如果你没有这个本领，可以直接测量实体模型并绘制图纸，就像第 11 章中所做的那样。（当然，如果你熟练掌握 CAD 工具，那么可以跳过制作实体模型这一步，直接绘制加工图纸。）

你可能还想将 Mega2560 上的 USB 连接器引出到背面板的 B 型连接器上。这样可以很方便地升级软件，获取发射运行数据，甚至把实时画面传送到远方的课堂供学生观看。

13.7.3 软件

对于这样一个设计研究，在软件方面我大致只能给你提供一些建议与几个框图，主要用意是帮你判断要投入多少精力编写软件支持的功能。如果软件完全根据功能需求设计并实现，那么应该可以直接将软件功能与功能需求一一映射。此外，不应该实现功能需求中未定义的功能。在工业，特别是航天工业中，软件一定不能包含功能需求未定义的功能。因为软件中那些实际并不需要的功能将得不到完全测试，未经完全测试的软件是十分危险的。

由于火箭发射器采用的是标准的 Arduino 类型设计，软件主模块必定包含 `setup()` 与 `loop()` 两个函数。此外，还需要一系列全局变量保存各种状态与时间数据。`loop()` 函数会执行一系列连续性的操作，比如监视系统状态、发射启动开关，根据需要打开或关闭 LED 灯，管理倒计时并在倒计时期间监视发射场安全开关。图 13-12 显示了主循环中要执行的操作。

在流程图 1 号部分，主循环启动时先读取旋转开关（点火器选择与计时选择）。如果从最后一次读取开关以后计时发生改变，并且激活了计数（比如正在发射），那么发射器将停止计数，取消系统待命状态并进入中止状态。

在 2 号部分，软件会查看待命开关。如果前一个循环迭代中待命开关是关闭的，而现在是打开的，则将系统待命标志设置为真，并打开担任主发射电源的继电器（见图 13-9）。如果系统待命标志为真，则检查是否正在进行计数。若是，则不会读发射启动开关（3 号部分），否则检查用户是否按下发射按钮。若按下，则将发射状态设置为真（on）并启动计数。

如果系统处于发射状态并且正在计数，控制流程就会转到图 13-12 的右手边执行。在 4 号部分，检查发射场安全开关是否按下（on），若没有则停止发射，让系统进入安全状态。

实际的发射动作主要发生在 5 号部分。当倒计时到 0 时，与所选点火电路相对应的发射继电器就会接通，继电器接通（处于 on 状态）2 s 后断开。这可能不是最好的方式，更好的做法是查看点火器状态，一旦所有点火器都打开就断开继电器。时间延迟是指出现故障之前最大可允许时间。

在 6 号部分中，要么发射成功，要么火箭还呆在发射架上（希望如此）。无论什么情形，系统都会恢复到等待状态，继电器断电，主电源继电器断开。我还建议你解除系统待命状态，这样用户必须将待命开关设置为 off，然后再设置到 on，让发射器重新进入待命状态。

读取开关输入、设置 LED 灯状态、读 RTC、控制继电器这一系列动作发生得很快。本设计看起来并不需要使用中断。话虽如此，你依然可能想为发射场安全开关使用中断。正因如此，前面才将发射场安全开关直接连接到 Mega2560 上，而不是 I/O 扩展板上。

发射场安全开关可以触发一个中断，当安全开关一放开，就立即断开为点火器提供电源的继电器（继电器模块 2 中的 3 号继电器）。即便 MCU 要花一点时间（几十 ms）读取开关状态并做出响应，让系统进入中止状态，火箭发动机点火器的电源也早已被切断。

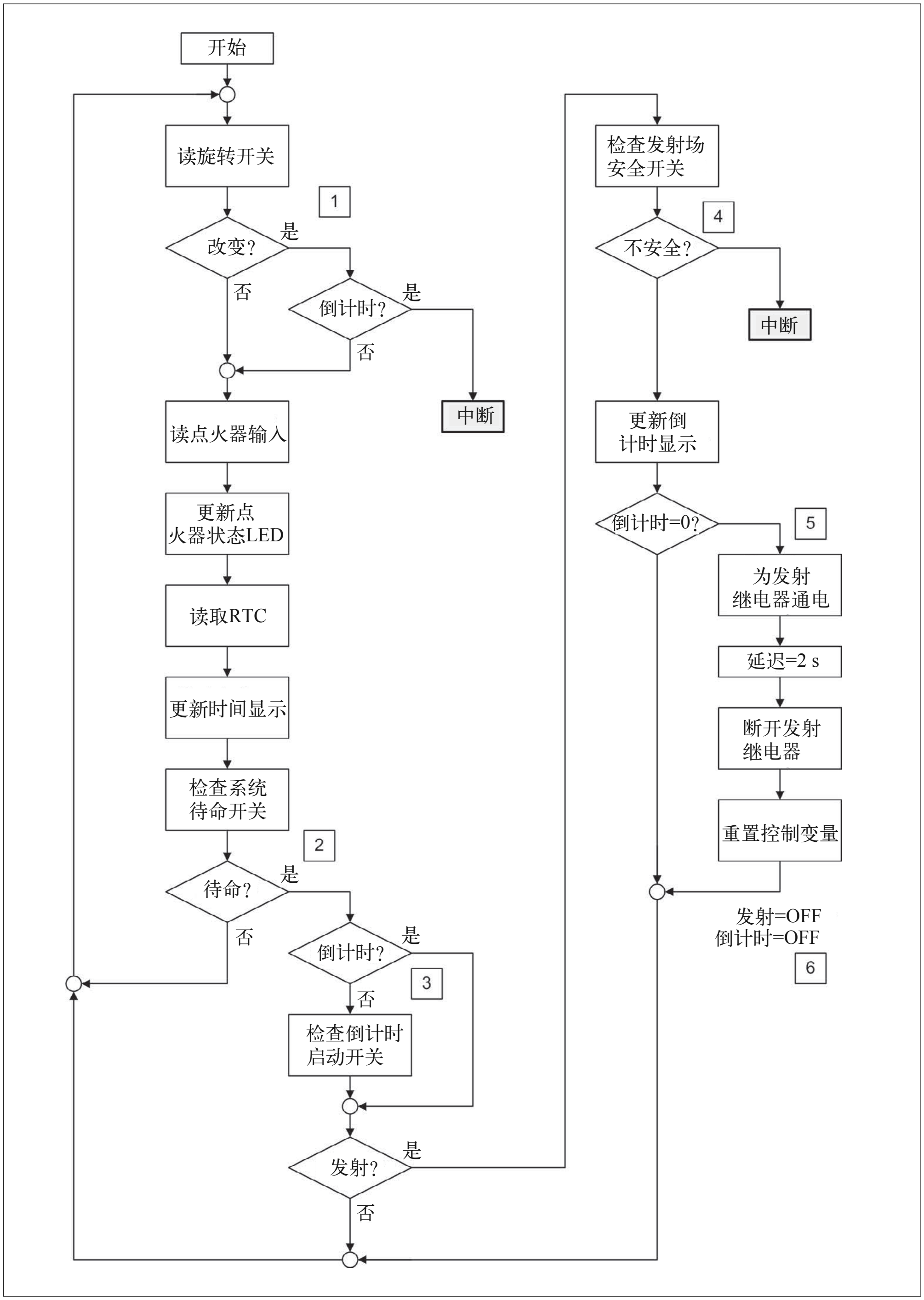


图 13-12：主循环中的操作

在循环运行期间，所有相关信息都会显示在 LCD 显示屏上。通过它能很容易看到系统的当前状态，发生故障时也能轻松获取相关信息。重复时间或倒计时时，使用 LCD 不是特别有用，它只会增加主循环的延迟（比如执行延迟）。

编写软件时，主要依据的是功能需求。对于项目列出的功能需求，软件必须予以实现，只有这样，编出的软件才符合需要，才能实现项目目标。关于使用模块化编写软件的方法，请参考第 10 章 ~ 第 12 章中的相关内容。

从图 13-12 中可以看到，软件由多个模块组成，点火器状态读取与更新是一个模块，RTC 与时间更新显示也是一个，主循环中的发射部分（流程图中 4 号与 6 号之间的部分）也可以编成一个单独的模块。RTC 模块、LCD、LED 显示器、I/O 扩展板最有可能定义为单独的类，但我们可能不需要为主循环的各种功能定义单独的类。其实它们并不复杂。

由于这是个设计分析项目而不是一个完整的设计，有关软件的介绍我只说这么多。如果你想继续进行，打算制作自己的火箭发射器，那么我建议你认真阅读前面 3 章的示例。以本章的设计分析为起点，写出其余细节应该不会有太多困难。

对于编写软件耗费的时间，我们无法预估，不能给出大致的结果。因为我们没有办法知道编写软件的人具体是谁。不同的人有不同的工作效率，有些人可能 1 h 就能写完代码，而有些人则可能需要大半天才能完成。所以，为了了解编写软件要耗费多长时间，我建议你尝试针对面包板或非专用原型（比如前面章节中见到的那些）编写一些测试代码，使用 RTC 或 LCD 做一些事情。了解大致要花费的时间之后，先把这个时间与软件设计包含的功能个数相乘，然后再乘以 2，最后算出的时间就是编写整个软件实际要花费的时间。如果你能在这个时间内写完软件，那就再好不过了。

13.7.4　测试与运行

不论哪种情况，有测试方案总归是好事。幸运的是，测试发射器相对容易，因为发射器本质上是由一些开关输入与继电器输出组成的。在点火器的输出接线柱之间简单连接跳线，即可对点火器的连续性进行模拟，然后将跳线去掉就变成了开路。

浏览功能需求会发现，它们都是可验证的。换言之，如果扭动点火器电路选择旋钮，你就会看到相应的 LED 灯从 1~6 依次点亮。如果将接线柱连接到模拟点火器，ICC GO 灯应该也会亮起。拨动开关，让系统进入待命状态，然后按住发射场安全开关，你将会看到 Safety Go 灯亮起。如果启动倒计时，然后断开任一激活的点火器，改变点火器电路或倒计时时间，或者释放发射场安全开关，系统将停止倒计时并进入中止状态。

在接线柱上使用跳线模拟点火器时，注意不要让倒计时走到 0。继电器闭合时，来自电池的全电流（full current）将流经跳线，这可能导致跳线熔断，或者损坏继电器，或者同时造成这两种结果。针对这一问题，一个更好的测试方案是使用可替换熔丝，或者在木板基座上安装实际点火器。此外，还可以使用小的白炽灯泡，要保证额定电压正确，并且有足够低的冷电阻（未点亮时）。标准的点火器电阻约为 0.5 Ω。

13.8 成本分析

制作发射器需要的成本取决于你在哪里购买电子元件，以及能否买到便宜的元件。其中，最主要成本支出是机箱壳，其价格可能超过 30 美元。你也可以选择使用 1 in × 1 in（2.5 cm × 2.5 cm）的硬木板与 1/4 in（0.6 cm）的纤维板，自己动手制作控制台，但前提是手上有必要的木工工具。购买所需材料并投入必要的时间后，你可能发现，还不如直接花钱买个现成的机箱更划算。

根据表 13-5 列出的元件，制作一个 Phase I 版的发射器的总费用大约为 125 美元，± 25 美元。而制作 Phase II 版的发射器的费用会远远超过 300 美元，但是通过货比三家，你可以降低总费用。

附录A

工具与配件

许多 Arduino 项目在制作时并不需要什么特别的工具，只要有一些跳线、扩展板与传感器 PCB，以及 Arduino 就够了。但当你开始学习制作更高级的项目时就会发现，拥有一系列基本工具与配件是很有必要的。通常，一套手工工具、一个焊枪、一些简单配件已足够满足大部分项目的需要，但对于高度复杂的项目，这些还不够。恕我冒昧，我建议你看一看我的另一本书《电子工程师必读：元器件与技术》，了解有关螺丝与螺栓大小、电子元器件、PCB 制作的内容。

本部分将介绍一些基本工具，你最好把它们放在手边。制作某个电子项目时，你会经常用到它们。如果不需要使用，那么可以很容易地把它们放进一个中等大小的工具箱里。

A.1　手工工具

一套好的手工工具是必不可少的。使用一套好的手工工具，再加上一点耐心与努力，你几乎可以完成任何任务。进入电气时代之前，手工工具是大多数人用来制造东西的唯一方法，他们也的确制造出大量令人惊叹的东西。只要愿意花时间认真做，你也能做出好的东西。此处不会讲解有关制作技术的内容，因为市面上已经有很多专门的图书了。我只介绍一些常用工具，并告诉你在哪里可以买到。

A.1.1　螺丝刀

对于大多数 Arduino 项目，你只要有一套小型螺丝刀或组合螺丝刀套装（如图 A-1 所示），以及一套大号螺丝刀就够了。在大多数货源充足的五金店、一些大型家居用品店，以及任何电子用品商店（电子元器件与电子工具专卖店），你都能看到各式各样的小型螺丝刀套装。



图 A-1：一套小型螺丝刀

图 A-2 所示的全尺寸螺丝刀很常见，比如大型杂货店的家用电器维修区。尽量不要买太大的工具，而要买带有小尖头的套装工具。实际制作中，你会经常用到这些小型工具，而大型工具则并不常用。（除非你需要连接住宅用电，或者制作与汽车有关的项目。）



图 A-2：一套标准的全尺寸螺丝刀

A.1.2　各种钳子

在电子项目制作中，尖嘴钳、斜口钳、一把好的剪线钳都是必需的。你可能还想要一把标准钳（standard pliers）或者电工钳（lineman’s pliers），但它们并非必不可少。图 A-3 显示的是一套基本的钳子与剪切钳，你可以成套购买，也可以根据自身需求单独购买。



图 A-3：一套基本的钳子与剪切钳

要抵挡住诱惑，不要购买五金店与家居用品商店捆绑销售的普通剪线钳。做 PCB 级的电子工作时也尽量不要使用它们，而要使用斜口剪线钳（flush cutter），它专门用于修剪元件引脚，剪切细导线。图 A-4 显示的是一把标准的斜口剪线钳。



图 A-4：电子制作中常用的斜口剪线钳

A.1.3　剥线钳

电子制作中，另一种必备工具是剥线钳。剥线时，虽然也可以选用剪线钳，但最好不要使用它，因为很容易产生误剪，导致把原本想保留的线缆剪断了。而且很容易在线芯上留下刮痕，使得导线在弯折时很容易断掉。我手头上有两把剥线钳，具体选用哪一把主要取决于要剥多少线，以及使用哪把最容易。

最简单的剥线钳由一对刀片组成，带有可调限位螺钉，如图 A-5 所示。其不便之处是，你每次为不同粗细的导线剥皮时，都得重新调整限位螺钉。但如果总是使用同一种导线，那么这并不是什么问题。（我几乎在所有电子项目中都选用 #24 的绝缘扭绞线，所以很少调整。）



图 A-5：最简单的剥线钳

我最喜欢的剥线钳来自 Klein，使用它们不但可以轻松为不同粗细的导线割断绝缘皮，还可以把这些绝缘皮剥离下来。图 A-6 显示的就是这样一种剥线钳。它们非常便宜，你可以购买更多切割刀片，以便为更多不同粗细的导线剥皮。这种剥线钳的缺点是，它们较大，使用起来有些笨拙，不适合狭小空间。

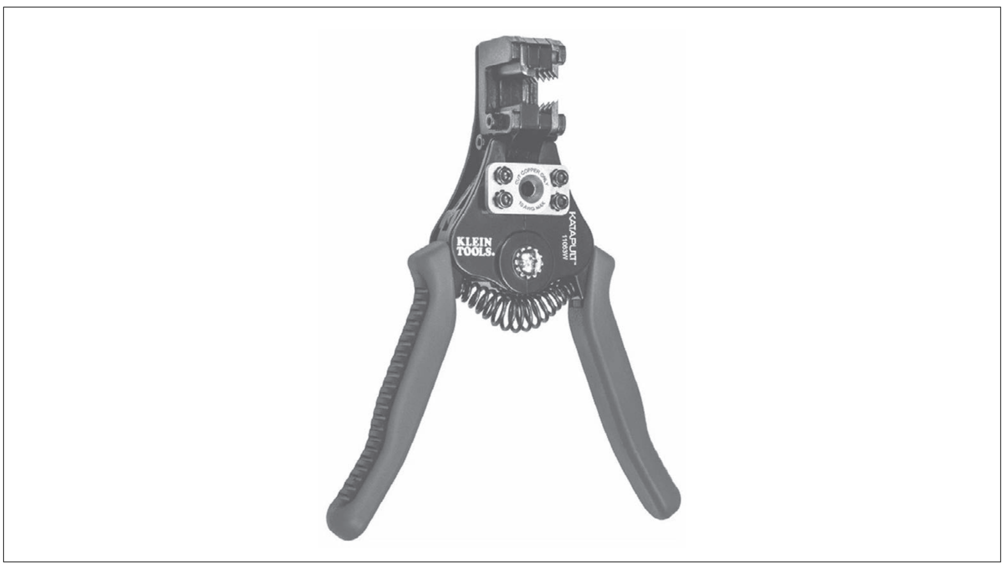


图 A-6：特别好用的剥线钳

A.1.4 连接器压接钳

使用 Arduino 开发板、各种扩展板、模块时，遇到的主要麻烦之一是连接其他电子器件。当然，你可以使用带有插脚或插口的跳线连接工作台（或厨桌）上的各种电子元件以搭建设备，并搞清其工作原理，但长期使用时，可靠性不佳。对于这个问题，一个更好的解决方法是使用 I/O 扩展板（比如第 9 章介绍的那些），这些扩展板提供的连接器都是多触点的。图 A-7 显示了一个带有多条连接线的扩展板。

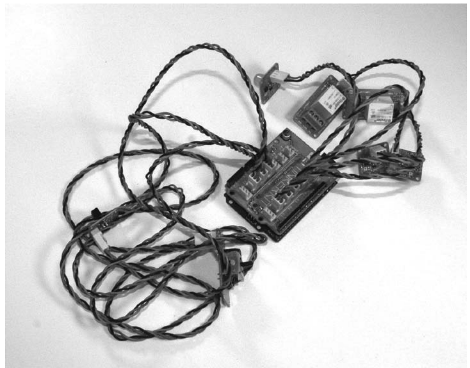


图 A-7：带有多条接线的 I/O 扩展板

金属制连接器都是通过压接方式连接至导线的，此时你需要一把特殊的工具——压接钳帮助完成这种连接（此外还需要连接器外壳与接插件）。令人欣慰的是，过去几年里，压接钳的价格大幅下降。现在只要花 30 美元左右就能买到一把压接钳，而在过去要花 200 美元。图 A-8 显示了各种压接钳。



图 A-8：各种低廉的压接钳

将（插针或插口）压接头压接到导线之后，接下来再将其装入连接器的壳套（又称外壳、壳体）。一个连接器壳套往往拥有多个插腔，这些插腔常见的间距为 0.1 in（2.54 mm），这是事实上的行业标准，在 Arduino 组件中很常见。图 A-9 显示的分别是 1 位、2 位、3 位、4 位的连接器壳套。连接器的插针或插口就埋在塑料壳套之中，借助一把小螺丝刀轻轻拉起锁闩，即可将连接器的插针或插口从壳套中轻松取出。

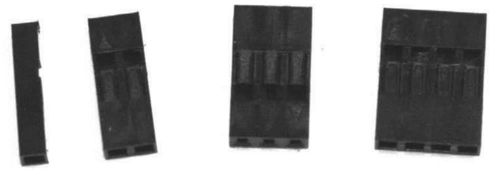


图 A-9：间距为 0.1 in（2.54 mm）的连接器壳套

一些 Arduino 扩展板与模块使用水晶头连接器连接其他元器件，类似于电话线或网线上使用的连接头。连接水晶头与线缆时，需要使用专门的工具，可以从大多数大型家居商场、电子元件分销商、网络销售商那里买到。图 A-10 显示的是一个使用这种连接器的扩展板。

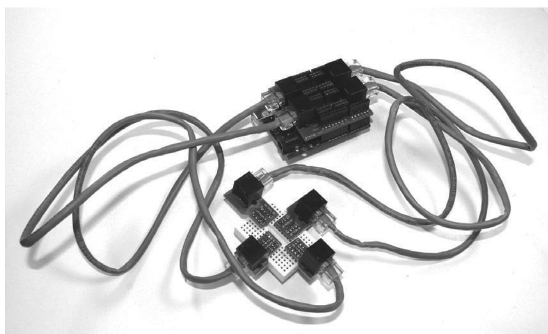


图 A-10：带有 RJ45 水晶头（8P8C）的 I/O 扩展板

最后，还有一种被称为“耳片接头”的连接器，它们广泛应用于电子系统与汽车，如图 A-11 所示。你可以很容易地购买到它们，但 Arduino 项目中并不常用（第 11 章的信号发生器中用到过）。这种连接器的类型和规格多种多样，相应的专用压接工具也可以从多种渠道购买。



图 A-11：带有扁平接线片的连接器

图 A-12 显示的就是一种专门压接“耳片接头”的工具。请不要使用这种压接工具对 PCB（比如 Arduino）上引脚与插口的小连接器进行压接，这样会导致连接器报废。



图 A-12：常见的扁平接线片压接工具

压接连接器连接容易、可靠（需要多练习才能熟练掌握），并且便宜。不足之处是，购买这些压接连接器只是第一步，此外还要购买专用的压接工具。如果愿意花这个钱并决定在项目中使用压接连接器，那么大部分时候你就可以不用焊枪了，你的项目看上去也会显得很优雅、美观、专业。

A.1.5　手锯

需要修整电路板、切割塑料外壳或塑料管时，手边有几把手锯是很方便的。做这些工作时，就切割速度与便利性而言，没有什么工具可以比得上手锯。

图 A-13 显示的是一把线锯，它在做精细切割时非常有用，但不适合切割大型物件。使用线锯的要领是，用它切割时不要用力按压（这也适用于其他手锯）。线锯的锯条很小，无法承受太大压力，但只有足够的细心与耐心，你几乎可以用它锯断任何东西。



图 A-13：线锯

对于切割较大的物体，特别是金属材料时，最好使用钢锯。图 A-14 显示的一种典型的钢锯。最新型的钢锯可能拥有更加流线化的外形，但基本结构都一样，主要由一根钢条与抓

握手柄组成。这种钢锯很常见，你可以从多个地方买到。

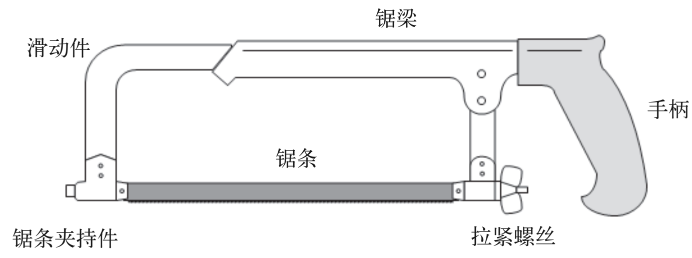


图 A-14：常见钢锯结构

使用钢锯时一定要注意，只能从一个方向切割物体——要么拉，要么推，具体取决于钢条的安装方向。安装锯条时，我喜欢把锯齿方向朝向手柄，这样锯割物体时只要拉锯即可。但有些人喜欢让锯齿朝着锯头方向，那样在锯物体时，他们必须向前推锯才行。

A.2　电动工具

对于有些工作，只要正确使用手动工具即可出色完成。但对于另外一些需要长期用力的工作，再使用手动工具就不合适了，这很容易引起肌肉痉挛，此时应该使用电动工具。电钻与电磨是最常用的两种电动工具。

A.2.1　电钻

手电钻用途广泛，但通常不适合精确打孔。当你需要在一个板子上打一个 1/8 in 的孔时，若不需要超级精准，那么手钻就非常有用。我建议你买一把图 A-15 所示的无绳电钻，主要是因为这种电钻没有讨厌的电源线，使用时会省去很多麻烦。虽然由电池供电的无绳电钻在动力方面可能比不上有线电钻（电源线直接插到墙上的电源插座），但由于大多数小项目中主要使用的是塑料、薄木板或仿木材料、薄金属等，使用无绳电钻不会有什么问题。



图 A-15：带有可更换电池的无绳电钻

A.2.2　小型研磨机

在电子制作中，虽然小型研磨机并非必不可少，但身边准备一台研磨机对于制作电子项目会非常有用、方便。图 A-16 显示的就是一台小型研磨机，你可以用它将改锥削尖，也可以打磨金属切割后留下的毛边与毛口，清理塑料的边缘，甚至修剪 PCB。

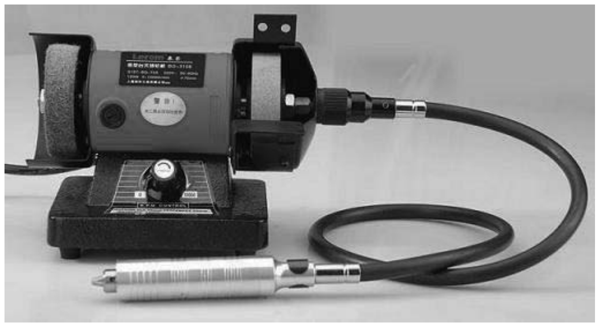


图 A-16：小型研磨机

上图这种特别的研磨机由 Harbor Freight 生产，它带有一个旋转头附件。使用该旋转附件虽然不能做电动雕刻机能做的所有工作，但对于一些轻量级的工作肯定能用得上。

A.2.3　小型钻床

如果需要在指定位置钻指定大小的孔，并且对精度有较高要求，那么你就需要使用钻床。当然页可以使用全尺寸钻床做这些工作，但这些全尺寸钻床都比较大，不使用时很难将其放入壁柜。解决方法是，购买图 A-17 所示的小型钻床。

除了在小塑料壳上为开关与 LED 灯钻安装孔之外，还可以用钻床在 PCB 上钻孔。另外，也可以购买一些辅助工具增加钻孔的稳定性，如小型台钳等。



图 A-17：小型钻床

A.3　焊接

焊接是电子电路的显著特征之一，但如果你在 Arduino 电子项目制作中使用现成的 PCB 与模块，那其实焊接也不是必须进行的。而如果你想将 Arduino 集成到一个更大的系统中，那就极有可能要用到焊接。如果碰巧购买了扩展板、一盒引脚与插口连接器，以及带有孔洞的 PCB，那么焊接就不再是可有可无的了。

A.3.1　电烙铁

电烙铁价格不一，有的只要几美元，这种超低价的电烙铁不带温度调节，烙铁头质量不高；有的则高达数百美元，这种电烙铁带有可更换的烙铁头，并可根据需要随意调节温度。请尽量不要买太便宜的电烙铁，它们质量不佳，可能会对电路板以及要焊接的元件造成严重损害。在自己财力允许的范围内，尽量买好的、贵的。即便手头很紧，也至少买一把图 A-18 所示的电烙铁，价格大约 15 美元。



图 A-18：便宜的电烙铁

如果你手头宽裕，建议购买一个图 A-19 所示的焊台（也叫“控温电烙铁”）。一般而言，焊台要比普通电烙铁贵一些，价格大约在 50~300 美元。买个好焊台是个不错的投资，但你得有点像样的活儿，这样才能对得起花的那笔钱。

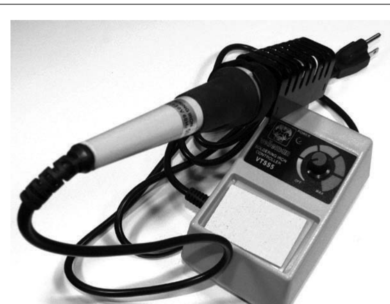


图 A-19：焊台

A.3.2 焊接辅件

买了电烙铁或焊台还不够，还要买一些基本的焊接辅件。如果缺了这些东西，电烙铁或焊台也没什么用。做焊接工作时，除了电烙铁，最起码还要有焊锡。请不要在本地五金店购买某些电子元件专用的锡焊，除非它明确指出也适用于焊接其他电子元件。好的电子级别（electronics-grade）的焊锡带有锡剂芯（一般是松香），大部分位于薄壁一侧。我喜欢买 454 g 大小的松脂芯焊丝，如图 A-20 所示。



图 A-20：松脂芯焊丝

其他常用的焊接辅件还有吸锡线（吸走多余焊锡的铜编织带）、液态或膏状助焊剂、焊锡膏。你可以在网上找到大量介绍焊接工具、焊接辅件、焊接技术的文章与视频，并从中学到更多知识。

A.4 工具购买渠道

关于上面介绍的这些工具，你可以从表 A-1 列出的供货商那里购买。其实，销售这些工具的供货商非常多，无法将其一一列出。表 A-1 只列出其中一部分，仅供各位参考。

表A-1：工具供应商

经销商/供应商	URL	经销商/供应商	URL
Adafruit	www.adafruit.com	Maker Shed	www.makershed.com
Apex Tool Group	www.apexhandtools.com	MCM Electronics	www.mcmelectronics.com
CKB Products	www.ckbproducts.com	SainSmart	www.sainsmart.com
Circuit Specialists	www.circuitspecialists.com	SparkFun	www.sparkfun.com
Electronic Goldmine	www.goldmine-elec-products.com	Stanley	www.stanleysupplyservices.com
Harbor Freight Tools	www.harborfreight.com	Velleman	www.vellemanusa.com

请不要忽视本地的二手工具店。许多城市都有一家或多家专卖二手工具的商店，里面的二手工具多种多样，从成桶的二手螺丝刀到车间机械设备（比如立式砂磨机），几乎无所不包。还有一些接收捐赠的慈善商店也值得逛一逛，比如 Goodwill（在美国这里）。或许里面可供选择的工具并不多，但如果有空还是可以去看看，说不定能碰到非常好的东西。

附录B

AVR ATmega控制寄存器

本部分不会全面讲解 AVR ATmega 的每一个控制寄存器，只对其中一些常用的做概括性介绍，以便各位快速查阅。有关特定 MCU 的每一个控制寄存器的详细介绍，请参考 Atmel 的相关技术文档。请特别注意 Atmel 技术文档有关控制寄存器概要中的注释。每个 MCU 中都有一些轻微不同需要留意。

一般来说，保留位（标记为 -）不应该被访问。0x00~0x1F 范围内的寄存器的位可以直接使用 SBI 与 CBI 指令（分别用于设置 I/O 位与清除 I/O 位）进行访问。括号中的寄存器地址是控制寄存器的 SRAM 地址，而不在括号中的寄存器地址存在于 64 位地址空间，专为 I/O 控制寄存器保留。预留地址可以在 IN 与 OUT 指令中使用，SRAM 地址必须使用 ST/STS/STD 与 LD/LDS/LDD 指令访问。

本部分内容来源于以下 Atmel 技术文档，你可以在 Atmel.com 网站上找到它们。

文档编号	标题
Atmel-8271I-AVR- ATmega-Datasheet_10/2014	Atmel ATmega48A/PA/88A/PA/168A/PA/328/P
2549Q-AVR-02/2014	Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V
7766F-AVR-11/10	Atmel ATmega16U4/ATmega32U4

B.1　ATmega 168/328

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
(0xFF)	保留	–	–	–	–	–	–	–	–
(0xFE)	保留	–	–	–	–	–	–	–	–
(0xFD)	保留	–	–	–	–	–	–	–	–
(0xFC)	保留	–	–	–	–	–	–	–	–

（续）

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
(0xFB)	保留	–	–	–	–	–	–	–	–
(0xFA)	保留	–	–	–	–	–	–	–	–
(0xF9)	保留	–	–	–	–	–	–	–	–
(0xF8)	保留	–	–	–	–	–	–	–	–
(0xF7)	保留	–	–	–	–	–	–	–	–
(0xF6)	保留	–	–	–	–	–	–	–	–
(0xF5)	保留	–	–	–	–	–	–	–	–
(0xF4)	保留	–	–	–	–	–	–	–	–
(0xF3)	保留	–	–	–	–	–	–	–	–
(0xF2)	保留	–	–	–	–	–	–	–	–
(0xF1)	保留	–	–	–	–	–	–	–	–
(0xF0)	保留	–	–	–	–	–	–	–	–
(0xEF)	保留	–	–	–	–	–	–	–	–
(0xEE)	保留	–	–	–	–	–	–	–	–
(0xED)	保留	–	–	–	–	–	–	–	–
(0xEC)	保留	–	–	–	–	–	–	–	–
(0xEB)	保留	–	–	–	–	–	–	–	–
(0xEA)	保留	–	–	–	–	–	–	–	–
(0xE9)	保留	–	–	–	–	–	–	–	–
(0xE8)	保留	–	–	–	–	–	–	–	–
(0xE7)	保留	–	–	–	–	–	–	–	–
(0xE6)	保留	–	–	–	–	–	–	–	–
(0xE5)	保留	–	–	–	–	–	–	–	–
(0xE4)	保留	–	–	–	–	–	–	–	–
(0xE3)	保留	–	–	–	–	–	–	–	–
(0xE2)	保留	–	–	–	–	–	–	–	–
(0xE1)	保留	–	–	–	–	–	–	–	–
(0xE0)	保留	–	–	–	–	–	–	–	–
(0xDF)	保留	–	–	–	–	–	–	–	–
(0xDE)	保留	–	–	–	–	–	–	–	–
(0xDD)	保留	–	–	–	–	–	–	–	–
(0xDC)	保留	–	–	–	–	–	–	–	–
(0xDB)	保留	–	–	–	–	–	–	–	–
(0xDA)	保留	–	–	–	–	–	–	–	–
(0xD9)	保留	–	–	–	–	–	–	–	–
(0xD8)	保留	–	–	–	–	–	–	–	–
(0xD7)	保留	–	–	–	–	–	–	–	–
(0xD6)	保留	–	–	–	–	–	–	–	–
(0xD5)	保留	–	–	–	–	–	–	–	–
(0xD4)	保留	–	–	–	–	–	–	–	–
(0xD3)	保留	–	–	–	–	–	–	–	–
(0xD2)	保留	–	–	–	–	–	–	–	–
(0xD1)	保留	–	–	–	–	–	–	–	–
(0xD0)	保留	–	–	–	–	–	–	–	–
(0xCF)	保留	–	–	–	–	–	–	–	–

（续）

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
(0xCE)	保留	–	–	–	–	–	–	–	–
(0xCD)	保留	–	–	–	–	–	–	–	–
(0xCC)	保留	–	–	–	–	–	–	–	–
(0xCB)	保留	–	–	–	–	–	–	–	–
(0xCA)	保留	–	–	–	–	–	–	–	–
(0xC9)	保留	–	–	–	–	–	–	–	–
(0xC8)	保留	–	–	–	–	–	–	–	–
(0xC7)	保留	–	–	–	–	–	–	–	–
(0xC6)	UDR0	USART	I/O 数据寄存器						
(0xC5)	UBRR0H	USART	波特率寄存器高位						
(0xC4)	UBRR0L	USART	波特率寄存器低位						
(0xC3)	保留	–	–	–	–	–	–	–	–
(0xC2)	UCSR0C	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01/ UDORD0	UCSZ00/ UCPHA0	UCPOL0
(0xC1)	UCSR0B	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80
(0xC0)	UCSR0A	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0
(0xBF)	保留	–	–	–	–	–	–	–	–
(0xBE)	保留	–	–	–	–	–	–	–	–
(0xBD)	TWAMR	TWAM6	TWAM5	TWAM4	TWAM3	TWAM2	TWAM1	TWAM0	–
(0xBC)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
(0xBB)	TWDR	双线串口数据寄存器							
(0xBA)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
(0xB9)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0
(0xB8)	TWBR	双线串口比特率寄存器							
(0xB7)	保留	–	–	–	–	–	–	–	–
(0xB6)	ASSR	–	EXCLK	AS2	TCN2UB	OCR2AUB	OCR2BUB	TCR2AUB	TCR2BUB
(0xB5)	保留	–	–	–	–	–	–	–	–
(0xB4)	OCR2B	时钟 / 计数器 2 输出比较寄存器 B							
(0xB3)	OCR2A	时钟 / 计数器 2 输出比较寄存器 A							
(0xB2)	TCNT2	时钟 / 计数器 2（8 位）							
(0xB1)	TCCR2B	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20
(0xB0)	TCCR2A	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20
(0xAF)	保留	–	–	–	–	–	–	–	–
(0xAE)	保留	–	–	–	–	–	–	–	–
(0xAD)	保留	–	–	–	–	–	–	–	–
(0xAC)	保留	–	–	–	–	–	–	–	–
(0xAB)	保留	–	–	–	–	–	–	–	–
(0xAA)	保留	–	–	–	–	–	–	–	–
(0xA9)	保留	–	–	–	–	–	–	–	–
(0xA8)	保留	–	–	–	–	–	–	–	–
(0xA7)	保留	–	–	–	–	–	–	–	–
(0xA6)	保留	–	–	–	–	–	–	–	–
(0xA5)	保留	–	–	–	–	–	–	–	–
(0xA4)	保留	–	–	–	–	–	–	–	–
(0xA3)	保留	–	–	–	–	–	–	–	–

（续）

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
(0xA2)	保留	–	–	–	–	–	–	–	–
(0xA1)	保留	–	–	–	–	–	–	–	–
(0xA0)	保留	–	–	–	–	–	–	–	–
(0x9F)	保留	–	–	–	–	–	–	–	–
(0x9E)	保留	–	–	–	–	–	–	–	–
(0x9D)	保留	–	–	–	–	–	–	–	–
(0x9C)	保留	–	–	–	–	–	–	–	–
(0x9B)	保留	–	–	–	–	–	–	–	–
(0x9A)	保留	–	–	–	–	–	–	–	–
(0x99)	保留	–	–	–	–	–	–	–	–
(0x98)	保留	–	–	–	–	–	–	–	–
(0x97)	保留	–	–	–	–	–	–	–	–
(0x96)	保留	–	–	–	–	–	–	–	–
(0x95)	保留	–	–	–	–	–	–	–	–
(0x94)	保留	–	–	–	–	–	–	–	–
(0x93)	保留	–	–	–	–	–	–	–	–
(0x92)	保留	–	–	–	–	–	–	–	–
(0x91)	保留	–	–	–	–	–	–	–	–
(0x90)	保留	–	–	–	–	–	–	–	–
(0x8F)	保留	–	–	–	–	–	–	–	–
(0x8E)	保留	–	–	–	–	–	–	–	–
(0x8D)	保留	–	–	–	–	–	–	–	–
(0x8C)	保留	–	–	–	–	–	–	–	–
(0x8B)	OCR1BH	时钟 / 计数器 1：输出比较寄存器 B 高字节							
(0x8A)	OCR1BL	时钟 / 计数器 1：输出比较寄存器 B 低字节							
(0x89)	OCR1AH	时钟 / 计数器 1：输出比较寄存器 A 高字节							
(0x88)	OCR1AL	时钟 / 计数器 1：输出比较寄存器 A 低字节							
(0x87)	ICR1H	时钟 / 计数器 1：输入捕获寄存器高字节							
(0x86)	ICR1L	时钟 / 计数器 1：输入捕获寄存器低字节							
(0x85)	TCNT1H	时钟 / 计数器 1：计数寄存器高字节							
(0x84)	TCNT1L	时钟 / 计数器 1：计数寄存器低字节							
(0x83)	保留	–	–	–	–	–	–	–	–
(0x82)	TCCR1C	FOC1A	FOC1B	–	–	–	–	–	–
(0x81)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
(0x80)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10
(0x7F)	DIDR1	–	–	–	–	–	–	AIN1D	AIN0D
(0x7E)	DIDR0	–	–	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D
(0x7D)	保留	–	–	–	–	–	–	–	–
(0x7C)	ADMUX	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0
(0x7B)	ADCSRB	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0
(0x7A)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
(0x79)	ADCH	ADC 数据寄存器高字节							
(0x78)	ADCL	ADC 数据寄存器低字节							
(0x77)	保留	–	–	–	–	–	–	–	–
(0x76)	保留	–	–	–	–	–	–	–	–

（续）

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
(0x75)	保留	–	–	–	–	–	–	–	–
(0x74)	保留	–	–	–	–	–	–	–	–
(0x73)	保留	–	–	–	–	–	–	–	–
(0x72)	保留	–	–	–	–	–	–	–	–
(0x71)	保留	–	–	–	–	–	–	–	–
(0x70)	TIMSK2	–	–	–	–	–	OCIE2B	OCIE2A	TOIE2
(0x6F)	TIMSK1	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1
(0x6E)	TIMSK0	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0
(0x6D)	PCMSK2	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16
(0x6C)	PCMSK1	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8
(0x6B)	PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0
(0x6A)	保留	–	–	–	–	–	–	–	–
(0x69)	EICRA	–	–	–	–	ISC11	ISC10	ISC01	ISC00
(0x68)	PCICR	–	–	–	–	–	PCIE2	PCIE1	PCIE0
(0x67)	保留	–	–	–	–	–	–	–	–
(0x66)	OSCCAL	振荡器校准寄存器							
(0x65)	保留	–	–	–	–	–	–	–	–
(0x64)	PRR	PRTWI	PRTIM2	PRTIM0	–	PRTIM1	PRSPI	PRUSART0	PRADC
(0x63)	保留	–	–	–	–	–	–	–	–
(0x62)	保留	–	–	–	–	–	–	–	–
(0x61)	CLKPR	CLKPCE	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0
(0x60)	WDTCSR	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C
0x3E (0x5E)	SPH	–	–	–	–	–	(SP10)	SP9	SP8
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
0x3C (0x5C)	保留	–	–	–	–	–	–	–	–
0x3B (0x5B)	保留	–	–	–	–	–	–	–	–
0x3A (0x5A)	保留	–	–	–	–	–	–	–	–
0x39 (0x59)	保留	–	–	–	–	–	–	–	–
0x38 (0x58)	保留	–	–	–	–	–	–	–	–
0x37 (0x57)	SPMCSR	SPMIE	(RWWSB)		–	(RWWSRE) BLBSET	PGWRT	PGERS	SELFPRGEN
0x36 (0x56)	保留	–	–	–	–	–	–	–	–
0x35 (0x55)	MCUCR	–	BODS	BODSE	PUD	–	–	IVSEL	IVCE
0x34 (0x54)	MCUSR	–	–	–	–	WDRF	BORF	EXTRF	PORF
0x33 (0x53)	SMCR	–	–	–	–	SM2	SM1	SM0	SE

（续）

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
0x32 (0x52)	保留	–	–	–	–	–	–	–	–
0x31 (0x51)	保留	–	–	–	–	–	–	–	–
0x30 (0x50)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
0x2F (0x4F)	保留	–	–	–	–	–	–	–	–
0x2E (0x4E)	SPDR	SPI 数据寄存器							
0x2D (0x4D)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X
0x2C (0x4C)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
0x2B (0x4B)	GPIOR2	通用 I/O 寄存器 2							
0x2A (0x4A)	GPIOR1	通用 I/O 寄存器 1							
0x29 (0x49)	保留	–	–	–	–	–	–	–	–
0x28 (0x48)	OCR0B	时钟 / 计数器 0 输出比较寄存器 B							
0x27 (0x47)	OCR0A	时钟 / 计数器 0 输出比较寄存器 A							
0x26 (0x46)	TCNT0	Timer/ Counter0	(8-bit)						
0x25 (0x45)	TCCR0B	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00
0x24 (0x44)	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00
0x23 (0x43)	GTCCR	TSM	–	–	–	–	–	PSRASY	PSRSYNC
0x22 (0x42)	EEARH	EEPROM 地址寄存器高字节							
0x21 (0x41)	EEARL	EEPROM 地址寄存器低字节							
0x20 (0x40)	EEDR	EEPROM 数据寄存器							
0x1F (0x3F)	EECR	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE
0x1E (0x3E)	GPIOR0	通用 I/O 寄存器 0							
0x1D (0x3D)	EIMSK	–	–	–	–	–	–	INT1	INT0
0x1C (0x3C)	EIFR	–	–	–	–	–	–	INTF1	INTF0
0x1B (0x3B)	PCIFR	–	–	–	–	–	PCIF2	PCIF1	PCIF0
0x1A (0x3A)	保留	–	–	–	–	–	–	–	–
0x19 (0x39)	保留	–	–	–	–	–	–	–	–

（续）

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
0x18 (0x38)	保留	–	–	–	–	–	–	–	–
0x17 (0x37)	TIFR2	–	–	–	–	–	OCF2B	OCF2A	TOV2
0x16 (0x36)	TIFR1	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1
0x15 (0x35)	TIFR0	–	–	–	–	–	OCF0B	OCF0A	TOV0
0x14 (0x34)	保留	–	–	–	–	–	–	–	–
0x13 (0x33)	保留	–	–	–	–	–	–	–	–
0x12 (0x32)	保留	–	–	–	–	–	–	–	–
0x11 (0x31)	保留	–	–	–	–	–	–	–	–
0x10 (0x30)	保留	–	–	–	–	–	–	–	–
0x0F (0x2F)	保留	–	–	–	–	–	–	–	–
0x0E (0x2E)	保留	–	–	–	–	–	–	–	–
0x0D (0x2D)	保留	–	–	–	–	–	–	–	–
0x0C (0x2C)	保留	–	–	–	–	–	–	–	–
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
0x08 (0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
0x07 (0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0x02 (0x22)	保留	–	–	–	–	–	–	–	–
0x01 (0x21)	保留	–	–	–	–	–	–	–	–
0x00 (0x20)	保留	–	–	–	–	–	–	–	–

B.2 ATmega 1280/2560

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
(0x1FF)	保留	–	–	–	–	–	–	–	–
...	保留	–	–	–	–	–	–	–	–
(0x137)	保留	–	–	–	–	–	–	–	–
(0x136)	UDR3	USART3 I/O 数据寄存器							
(0x135)	UBRR3H	–	–	–	–	USART3 波特率寄存器高字节			
(0x134)	UBRR3L	USART3 波特率寄存器低字节							
(0x133)	保留	–	–	–	–	–	–	–	–
(0x132)	UCSR3C	UMSEL31	UMSEL30	UPM31	UPM30	USBS3	UCSZ31	UCSZ30	UCPOL3
(0x131)	UCSR3B	RXCIE3	TXCIE3	UDRIE3	RXEN3	TXEN3	UCSZ32	RXB83	TXB83
(0x130)	UCSR3A	RXC3	TXC3	UDRE3	FE3	DOR3	UPE3	U2X3	MPCM3
(0x12F)	保留	–	–	–	–	–	–	–	–
(0x12E)	保留	–	–	–	–	–	–	–	–
(0x12D)		时钟 / 计数器 5：输出比较寄存器 C 高字节							
(0x12C)		时钟 / 计数器 5：输出比较寄存器 C 低字节							
(0x12B)		时钟 / 计数器 5：输出比较寄存器 B 高字节							
(0x12A)		时钟 / 计数器 5：输出比较寄存器 B 低字节							
(0x129)		时钟 / 计数器 5：输出比较寄存器 A 高字节							
(0x128)		时钟 / 计数器 5：输出比较寄存器 A 低字节							
(0x127)		时钟 / 计数器 5：输入捕获寄存器高字节							
(0x126)		时钟 / 计数器 5：输入捕获寄存器低字节							
(0x125)		时钟 / 计数器 5：计数寄存器高字节							
(0x124)		时钟 / 计数器 5：计数器寄存器低字节							
(0x123)	保留	–	–	–	–	–	–	–	–
(0x122)	TCCR5C	FOC5A	FOC5B	FOC5C	–	–	–	–	–
(0x121)	TCCR5B	ICNC5	ICES5	–	WGM53	WGM52	CS52	CS51	CS50
(0x120)	TCCR5A	COM5A1	COM5A0	COM5B1	COM5B0	COM5C1	COM5C0	WGM51	WGM50
(0x11F)	保留	–	–	–	–	–	–	–	–
(0x11E)	保留	–	–	–	–	–	–	–	–
(0x11D)	保留	–	–	–	–	–	–	–	–
(0x11C)	保留	–	–	–	–	–	–	–	–
(0x11B)	保留	–	–	–	–	–	–	–	–
(0x11A)	保留	–	–	–	–	–	–	–	–
(0x119)	保留	–	–	–	–	–	–	–	–
(0x118)	保留	–	–	–	–	–	–	–	–
(0x117)	保留	–	–	–	–	–	–	–	–
(0x116)	保留	–	–	–	–	–	–	–	–
(0x115)	保留	–	–	–	–	–	–	–	–
(0x114)	保留	–	–	–	–	–	–	–	–
(0x113)	保留	–	–	–	–	–	–	–	–
(0x112)	保留	–	–	–	–	–	–	–	–
(0x111)	保留	–	–	–	–	–	–	–	–

（续）

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
(0x110)	保留	–	–	–	–	–	–	–	–
(0x10F)	保留	–	–	–	–	–	–	–	–
(0x10E)	保留	–	–	–	–	–	–	–	–
(0x10D)	保留	–	–	–	–	–	–	–	–
(0x10C)	保留	–	–	–	–	–	–	–	–
(0x10B)	PORTL	PORTL7	PORTL6	PORTL5	PORTL4	PORTL3	PORTL2	PORTL1	PORTL0
(0x10A)	DDRL	DDL7	DDL6	DDL5	DDL4	DDL3	DDL2	DDL1	DDL0
(0x109)	PINL	PINL7	PINL6	PINL5	PINL4	PINL3	PINL2	PINL1	PINL0
(0x108)	PORTK	PORTK7	PORTK6	PORTK5	PORTK4	PORTK3	PORTK2	PORTK1	PORTK0
(0x107)	DDRK	DDK7	DDK6	DDK5	DDK4	DDK3	DDK2	DDK1	DDK0
(0x106)	PINK	PINK7	PINK6	PINK5	PINK4	PINK3	PINK2	PINK1	PINK0
(0x105)	PORTJ	PORTJ7	PORTJ6	PORTJ5	PORTJ4	PORTJ3	PORTJ2	PORTJ1	PORTJ0
(0x104)	DDRJ	DDJ7	DDJ6	DDJ5	DDJ4	DDJ3	DDJ2	DDJ1	DDJ0
(0x103)	PINJ	PINJ7	PINJ6	PINJ5	PINJ4	PINJ3	PINJ2	PINJ1	PINJ0
(0x102)	PORTH	PORTH7	PORTH6	PORTH5	PORTH4	PORTH3	PORTH2	PORTH1	PORTH0
(0x101)	DDRH	DDH7	DDH6	DDH5	DDH4	DDH3	DDH2	DDH1	DDH0
(0x100)	PINH	PINH7	PINH6	PINH5	PINH4	PINH3	PINH2	PINH1	PINH0
(0xFF)	保留	–	–	–	–	–	–	–	–
(0xFE)	保留	–	–	–	–	–	–	–	–
(0xFD)	保留	–	–	–	–	–	–	–	–
(0xFC)	保留	–	–	–	–	–	–	–	–
(0xFB)	保留	–	–	–	–	–	–	–	–
(0xFA)	保留	–	–	–	–	–	–	–	–
(0xF9)	保留	–	–	–	–	–	–	–	–
(0xF8)	保留	–	–	–	–	–	–	–	–
(0xF7)	保留	–	–	–	–	–	–	–	–
(0xF6)	保留	–	–	–	–	–	–	–	–
(0xF5)	保留	–	–	–	–	–	–	–	–
(0xF4)	保留	–	–	–	–	–	–	–	–
(0xF3)	保留	–	–	–	–	–	–	–	–
(0xF2)	保留	–	–	–	–	–	–	–	–
(0xF1)	保留	–	–	–	–	–	–	–	–
(0xF0)	保留	–	–	–	–	–	–	–	–
(0xEF)	保留	–	–	–	–	–	–	–	–
(0xEE)	保留	–	–	–	–	–	–	–	–
(0xED)	保留	–	–	–	–	–	–	–	–
(0xEC)	保留	–	–	–	–	–	–	–	–
(0xEB)	保留	–	–	–	–	–	–	–	–
(0xEA)	保留	–	–	–	–	–	–	–	–
(0xE9)	保留	–	–	–	–	–	–	–	–
(0xE8)	保留	–	–	–	–	–	–	–	–
(0xE7)	保留	–	–	–	–	–	–	–	–

（续）

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
(0xE6)	保留	–	–	–	–	–	–	–	–
(0xE5)	保留	–	–	–	–	–	–	–	–
(0xE4)	保留	–	–	–	–	–	–	–	–
(0xE3)	保留	–	–	–	–	–	–	–	–
(0xE2)	保留	–	–	–	–	–	–	–	–
(0xE1)	保留	–	–	–	–	–	–	–	–
(0xE0)	保留	–	–	–	–	–	–	–	–
(0xDF)	保留	–	–	–	–	–	–	–	–
(0xDE)	保留	–	–	–	–	–	–	–	–
(0xDD)	保留	–	–	–	–	–	–	–	–
(0xDC)	保留	–	–	–	–	–	–	–	–
(0xDB)	保留	–	–	–	–	–	–	–	–
(0xDA)	保留	–	–	–	–	–	–	–	–
(0xD9)	保留	–	–	–	–	–	–	–	–
(0xD8)	保留	–	–	–	–	–	–	–	–
(0xD7)	保留	–	–	–	–	–	–	–	–
(0xD6)	UDR2	USART I/O 数据寄存器							
(0xD5)	UBRR2H	–	–	–	–	USART2 波特率寄存器高字节			
(0xD4)	UBRR2L	USART2 波特率寄存器低字节							
(0xD3)	保留	–	–	–	–	–	–	–	–
(0xD2)	UCSR2C	UMSEL21	UMSEL20	UPM21	UPM20	USBS2	UCSZ21	UCSZ20	UCPOL2
(0xD1)	UCSR2B	RXCIE2	TXCIE2	UDRIE2	RXEN2	TXEN2	UCSZ22	RXB82	TXB82
(0xD0)	UCSR2A	RXC2	TXC2	UDRE2	FE2	DOR2	UPE2	U2X2	MPCM2
(0xCF)	保留	–	–	–	–	–	–	–	–
(0xCE)	UDR1	USART1 I/O 数据寄存器							
(0xCD)	UBRR1H	–	–	–	–	USART1 波特率寄存器高字节			
(0xCC)	UBRR1L	USART1 波特率寄存器低字节							
(0xCB)	保留	–	–	–	–	–	–	–	–
(0xCA)	UCSR1C	UMSEL11	UMSEL10	UPM11	UPM10	USBS1	UCSZ11	UCSZ10	UCPOL1
(0xC9)	UCSR1B	RXCIE1	TXCIE1	UDRIE1	RXEN1	TXEN1	UCSZ12	RXB81	TXB81
(0xC8)	UCSR1A	RXC1	TXC1	UDRE1	FE1	DOR1	UPE1	U2X1	MPCM1
(0xC7)	保留	–	–	–	–	–	–	–	–
(0xC6)	UDR0	USART0 I/O 数据寄存器							
(0xC5)	UBRR0H	–	–	–	–	USART 波特率寄存器高字节			
(0xC4)	UBRR0L	USART0 波特率寄存器低字节							
(0xC3)	保留	–	–	–	–	–	–	–	–
(0xC2)	UCSR0C	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01	UCSZ00	UCPOL0
(0xC1)	UCSR0B	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80
(0xC0)	UCSR0A	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0
(0xBF)	保留	–	–	–	–	–	–	–	–
(0xBE)	保留	–	–	–	–	–	–	–	–
(0xBD)	TWAMR	TWAM6	TWAM5	TWAM4	TWAM3	TWAM2	TWAM1	TWAM0	-

（续）

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
(0xBC)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE
(0xBB)	TWDR	双线串口位速率寄存器							
(0xBA)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
(0xB9)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0
(0xB8)	TWBR	双线串口比特率寄存器							
(0xB7)	保留	–	–	–	–	–	–	–	–
(0xB6)	ASSR	–	EXCLK	AS2	TCN2UB	OCR2AUB	OCR2BUB	TCR2AUB	TCR2BUB
(0xB5)	保留	–	–	–	–	–	–	–	–
(0xB4)	OCR2B	时钟 / 计数器 2 输出比较寄存器 B							
(0xB3)	OCR2A	时钟 / 计数器 2 输出比较寄存器 A							
(0xB2)	TCNT2	时钟 / 计数器 2（8 位）							
(0xB1)	TCCR2B	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20
(0xB0)	TCCR2A	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20
(0xAF)	保留	–	–	–	–	–	–	–	–
(0xAE)	保留	–	–	–	–	–	–	–	–
(0xAD)	OCR4CH	时钟 / 计数器 4：输出比较寄存器 C 高字节							
(0xAC)	OCR4CL	时钟 / 计数器 4：输出比较寄存器 C 低字节							
(0xAB)	OCR4BH	时钟 / 计数器 4：输出比较寄存器 B 高字节							
(0xAA)	OCR4BL	时钟 / 计数器 4：输出比较寄存器 B 低字节							
(0xA9)	OCR4AH	时钟 / 计数器 4：输出比较寄存器 A 高字节							
(0xA8)	OCR4AL	时钟 / 计数器 4：输出比较寄存器 A 低字节							
(0xA7)	ICR4H	时钟 / 计数器 4：输入捕获寄存器高字节							
(0xA6)	ICR4L	时钟 / 计数器 4：输入捕获寄存器低字节							
(0xA5)	TCNT4H	时钟 / 计数器 4：计数器寄存器高字节							
(0xA4)	TCNT4L	时钟 / 计数器 4：计数器寄存器低字节							
(0xA3)	保留	–	–	–	–	–	–	–	–
(0xA2)	TCCR4C	FOC4A	FOC4B	FOC4C	–	–	–	–	–
(0xA1)	TCCR4B	ICNC4	ICES4	–	WGM43	WGM42	CS42	CS41	CS40
(0xA0)	TCCR4A	COM4A1	COM4A0	COM4B1	COM4B0	COM4C1	COM4C0	WGM41	WGM40
(0x9F)	保留	–	–	–	–	–	–	–	–
(0x9E)	保留	–	–	–	–	–	–	–	–
(0x9D)	OCR3CH	时钟 / 计数器 3：输出比较寄存器 C 高字节							
(0x9C)	OCR3CL	时钟 / 计数器 3：输出比较寄存器 C 低字节							
(0x9B)	OCR3BH	时钟 / 计数器 3：输出比较寄存器 B 高字节							
(0x9A)	OCR3BL	时钟 / 计数器 3：输出比较寄存器 B 低字节							
(0x99)	OCR3AH	时钟 / 计数器 3：输出比较寄存器 A 高字节							
(0x98)	OCR3AL	时钟 / 计数器 3：输出比较寄存器 A 低字节							
(0x97)	ICR3H	时钟 / 计数器 3：输入捕获寄存器高字节							
(0x96)	ICR3L	时钟 / 计数器 3：输入捕获寄存器低字节							
(0x95)	TCNT3H	时钟 / 计数器 3：计数器寄存器高字节							
(0x94)	TCNT3L	时钟 / 计数器 3：计数器寄存器低字节							
(0x93)	保留	–	–	–	–	–	–	–	–

（续）

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
(0x92)	TCCR3C	FOC3A	FOC3B	FOC3C	–	–	–	–	–
(0x91)	TCCR3B	ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30
(0x90)	TCCR3A	COM3A1	COM3A0	COM3B1	COM3B0	COM3C1	COM3C0	WGM31	WGM30
(0x8F)	保留	–	–	–	–	–	–	–	–
(0x8E)	保留	–	–	–	–	–	–	–	–
(0x8D)	OCR1CH	时钟 / 计数器 1：输出比较寄存器 C 高字节							
(0x8C)	OCR1CL	时钟 / 计数器 1：输出比较寄存器 C 低字节							
(0x8B)	OCR1BH	时钟 / 计数器 1：输出比较寄存器 B 高字节							
(0x8A)	OCR1BL	时钟 / 计数器 1：输出比较寄存器 B 低字节							
(0x89)	OCR1AH	时钟 / 计数器 1：输出比较寄存器 A 高字节							
(0x88)	OCR1AL	时钟 / 计数器 1：输出比较寄存器 A 低字节							
(0x87)	ICR1H	时钟 / 计数器 1：输入捕获寄存器高字节							
(0x86)	ICR1L	时钟 / 计数器 1：输入捕获寄存器低字节							
(0x85)	TCNT1H	时钟 / 计数器 1：计数器寄存器高字节							
(0x84)	TCNT1L	时钟 / 计数器 1：计数器寄存器低字节							
(0x83)	保留	–	–	–	–	–	–	–	–
(0x82)	TCCR1C	FOC1A	FOC1B	FOC1C	–	–	–	–	–
(0x81)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
(0x80)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10
(0x7F)	DIDR1	–	–	–	–	–	–	AIN1D	AIN0D
(0x7E)	DIDR0	ADC7D	ADC6D	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D
(0x7D)	DIDR2	ADC15D	ADC14D	ADC13D	ADC12D	ADC11D	ADC10D	ADC9D	ADC8D
(0x7C)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
(0x7B)	ADCSRB	–	ACME	–	–	MUX5	ADTS2	ADTS1	ADTS0
(0x7A)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
(0x79)	ADCH	ADC 数据寄存器高字节							
(0x78)	ADCL	ADC 数据寄存器低字节							
(0x77)	保留	–	–	–	–	–	–	–	–
(0x76)	保留	–	–	–	–	–	–	–	–
(0x75)	XMCRB	XMBK	–	–	–	–	XMM2	XMM1	XMM0
(0x74)	XMCRA	SRE	SRL2	SRL1	SRL0	SRW11	SRW10	SRW01	SRW00
(0x73)	TIMSK5	–	–	ICIE5	–	OCIE5C	OCIE5B	OCIE5A	TOIE5
(0x72)	TIMSK4	–	–	ICIE4	–	OCIE4C	OCIE4B	OCIE4A	TOIE4
(0x71)	TIMSK3	–	–	ICIE3	–	OCIE3C	OCIE3B	OCIE3A	TOIE3
(0x70)	TIMSK2	–	–	–	–	–	OCIE2B	OCIE2A	TOIE2
(0x6F)	TIMSK1	–	–	ICIE1	–	OCIE1C	OCIE1B	OCIE1A	TOIE1
(0x6E)	TIMSK0	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0
(0x6D)	PCMSK2	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16
(0x6C)	PCMSK1	PCINT15	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8
(0x6B)	PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0
(0x6A)	EICRB	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40
(0x69)	EICRA	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00

（续）

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
(0x68)	PCICR	–	–	–	–	–	PCIE2	PCIE1	PCIE0
(0x67)	保留	–	–	–	–	–	–	–	–
(0x66)	OSCCAL	振荡器校准寄存器							
(0x65)	PRR1	–	–	PRTIM5	PRTIM4	PRTIM3	PRUSART3	PRUSART2	PRUSART1
(0x64)	PRR0	PRTWI	PRTIM2	PRTIM0	–	PRTIM1	PRSPI	PRUSART0	PRADC
(0x63)	保留	–	–	–	–	–	–	–	–
(0x62)	保留	–	–	–	–	–	–	–	–
(0x61)	CLKPR	CLKPCE	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0
(0x60)	WDTCSR	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C
0x3E (0x5E)	SPH	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
0x3C (0x5C)	EIND	–	–	–	–	–	–	–	EIND0
0x3B (0x5B)	RAMPZ	–	–	–	–	–	–	RAMPZ1	RAMPZ0
0x3A (0x5A)	保留	–	–	–	–	–	–	–	–
0x39 (0x59)	保留	–	–	–	–	–	–	–	–
0x38 (0x58)	保留	–	–	–	–	–	–	–	–
0x37 (0x57)	SPMCSR	SPMIE	RWWSB	SIGRD	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN
0x36 (0x56)	保留	–	–	–	–	–	–	–	–
0x35 (0x55)	MCUCR	JTD	–	–	PUD	–	–	IVSEL	IVCE
0x34 (0x54)	MCUSR	–	–	–	JTRF	WDRF	BORF	EXTRF	PORF
0x33 (0x53)	SMCR	–	–	–	–	SM2	SM1	SM0	SE
0x32 (0x52)	保留	–	–	–	–	–	–	–	–
0x31 (0x51)	OCDR	OCDR7	OCDR6	OCDR5	OCDR4	OCDR3	OCDR2	OCDR1	OCDR0
0x30 (0x50)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
0x2F (0x4F)	保留	–	–	–	–	–	–	–	–
0x2E (0x4E)	SPDR	SPI 数据寄存器							
0x2D (0x4D)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X

（续）

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
0x2C (0x4C)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
0x2B (0x4B)	GPIOR2	通用 I/O 寄存器 2							
0x2A (0x4A)	GPIOR1	通用 I/O 寄存器 1							
0x29 (0x49)	保留	–	–	–	–	–	–	–	–
0x28 (0x48)	OCR0B	时钟 / 计数器 0 输出比较寄存器 B							
0x27 (0x47)	OCR0A	时钟 / 计数器 0 输出比较寄存器 A							
0x26 (0x46	TCNT0	时钟 / 计数器 0（8 位）							
0x25 (0x45)	TCCR0B	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00
0x24 (0x44)	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00
0x23 (0x43)	GTCCR	TSM	–	–	–	–	–	PSRASY	PSRSYNC
0x22 (0x42)	EEARH	–	–	–	–	EEPROM 地址寄存器高字节			
0x21 (0x41)	EEARL	EEPROM 地址寄存器低字节							
0x20 (0x40)	EEDR	EEPROM 数据寄存器							
0x1F (0x3F)	EECR	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE
0x1E (0x3E)	GPIOR0	通用 I/O 寄存器 0							
0x1D (0x3D)	EIMSK	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0
0x1C (0x3C)	EIFR	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	INTF0
0x1B (0x3B)	PCIFR	–	–	–	–	–	PCIF2	PCIF1	PCIF0
0x1A (0x3A)	TIFR5	–	–	ICF5	–	OCF5C	OCF5B	OCF5A	TOV5
0x19 (0x39)	TIFR4	–	–	ICF4	–	OCF4C	OCF4B	OCF4A	TOV4
0x18 (0x38)	TIFR3	–	–	ICF3	–	OCF3C	OCF3B	OCF3A	TOV3
0x17 (0x37)	TIFR2	–	–	–	–	–	OCF2B	OCF2A	TOV2
0x16 (0x36)	TIFR1	–	–	ICF1	–	OCF1C	OCF1B	OCF1A	TOV1
0x15 (0x35)	TIFR0	–	–	–	–	–	OCF0B	OCF0A	TOV0

（续）

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
0x14 (0x34)	PORTG	–	–	PORTG5	PORTG4	PORTG3	PORTG2	PORTG1	PORTG0
0x13 (0x33)	DDRG	–	–	DDG5	DDG4	DDG3	DDG2	DDG1	DDG0
0x12 (0x32)	PING	–	–	PING5	PING4	PING3	PING2	PING1	PING0
0x11 (0x31)	PORTF	PORTF7	PORTF6	PORTF5	PORTF4	PORTF3	PORTF2	PORTF1	PORTF0
0x10 (0x30)	DDRF	DDF7	DDF6	DDF5	DDF4	DDF3	DDF2	DDF1	DDF0
0x0F (0x2F)	PINF	PINF7	PINF6	PINF5	PINF4	PINF3	PINF2	PINF1	PINF0
0x0E (0x2E)	PORTE	PORTE7	PORTE6	PORTE5	PORTE4	PORTE3	PORTE2	PORTE1	PORTE0
0x0D (0x2D)	DDRE	DDE7	DDE6	DDE5	DDE4	DDE3	DDE2	DDE1	DDE0
0x0C (0x2C)	PINE	PINE7	PINE6	PINE5	PINE4	PINE3	PINE2	PINE1	PINE0
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
0x08 (0x28)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
0x07 (0x27)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
0x06 (0x26)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0x02 (0x22)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
0x01 (0x21)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
0x00 (0x20)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0

B.3 ATmega32U4

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
(0xFF)	Reserved	–	–	–	–	–	–	–	–
(0xFE)	Reserved	–	–	–	–	–	–	–	–
(0xFD)	Reserved	–	–	–	–	–	–	–	–
(0xFC)	Reserved	–	–	–	–	–	–	–	–
(0xFB)	Reserved	–	–	–	–	–	–	–	–
(0xFA)	Reserved	–	–	–	–	–	–	–	–
(0xF9)	Reserved	–	–	–	–	–	–	–	–
(0xF8)	Reserved	–	–	–	–	–	–	–	–
(0xF7)	Reserved	–	–	–	–	–	–	–	–
(0xF6)	Reserved	–	–	–	–	–	–	–	–
(0xF5)	Reserved	–	–	–	–	–	–	–	–
(0xF4)	UEINT	–	EPINT6:0						
(0xF3)	UEBCHX	–	–	–	–	–	BYCT10:8		
(0xF2)	UEBCLX	BYCT7:0							
(0xF1)	UEDATX	DAT7:0							
(0xF0)	UEIENX	FLERRE	NAKINE	–	NAKOUTE	RXSTPE	RXOUTE	STALLEDE	TXINE
(0xEF)	UESTA1X	–	–	–	–	–	CTRLDIR	CURRBK1:0	
(0xEE)	UESTA0X	CFGOK	OVERFI	UNDERFI	–	DTSEQ1:0		NBUSYBK1:0	
(0xED)	UECFG1X	–	EPSIZE2:0			EPBK1:0		ALLOC	–
(0xEC)	UECFG0X	EPTYPE1:0		–	–	–	–	–	EPDIR
(0xEB)	UECONX	–	–	STALLRQ	STALLRQC	RSTDT	–	–	EPEN
(0xEA)	UERST	–	EPRST6:0						
(0xE9)	UENUM	–	–	–	–	–	EPNUM2:0		
(0xE8)	UEINTX	FIFOCON	NAKINI	RWAL	NAKOUTI	RXSTPI	RXOUTI	STALLEDI	TXINI
(0xE7)	Reserved	–	–	–	–	–	–	–	–
(0xE6)	UDMFN	–	–	–	FNCERR	–	–	–	–
(0xE5)	UDFNUMH	–	–	–	–	–	FNUM10:8		
(0xE4)	UDFNUML	FNUM7:0							
(0xE3)	UDADDR	ADDEN	UADD6:0						
(0xE2)	UDIEN	–	UPRSME	EORSME	WAKEUPE	EORSTE	SOFE	MSOFE	SUSPE
(0xE1)	UDINT	–	UPRSMI	EORSMI	WAKEUPI	EORSTI	SOFI	MSOFI	SUSPI
(0xE0)	UDCON	–	–	–	–	RSTCPU	LSM	RMWKUP	DETACH
(0xDF)	Reserved	–	–	–	–	–	–	–	–
(0xDE)	Reserved	–	–	–	–	–	–	–	–
(0xDD)	Reserved	–	–	–	–	–	–	–	–
(0xDC)	Reserved	–	–	–	–	–	–	–	–
(0xDB)	Reserved	–	–	–	–	–	–	–	–
(0xDA)	USBINT	–	–	–	–	–	–	–	VBUSTI
(0xD9)	USBSTA	–	–	–	–	–	–	ID	VBUS
(0xD8)	USBCON	USBE	–	FRZCLK	OTGPADE	–	–	–	VBUSTE
(0xD7)	UHWCON	–	–	–	–	–	–	–	UVREGE
(0xD6)	Reserved	–	–	–	–	–	–	–	–
(0xD5)	Reserved	–	–	–	–	–	–	–	–

（续）

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
(0xD4)	DT4	DT4H3	DT4H2	DT4H1	DT4H0	DT4L3	DT4L2	DT4L1	DT4L0
(0xD3)	Reserved	–	–	–	–	–	–	–	–
(0xD2)	OCR4D	时钟 / 计数器 4：输出比较寄存器 D							
(0xD1)	OCR4C	时钟 / 计数器 4：输出比较寄存器 C							
(0xD0)	OCR4B	时钟 / 计数器 4：输出比较寄存器 B							
(0xCF)	OCR4A	时钟 / 计数器 4：输出比较寄存器 A							
(0xCE)	UDR1	USART1 I/O 数据寄存器							
(0xCD)	UBRR1H	–	–	–	–	USART1 波特率寄存器高字节			
(0xCC)	UBRR1L	USART1 波特率寄存器低字节							
(0xCB)	UCSR1D	–	–	–	–	–	–	CTSEN	RTSEN
(0xCA)	UCSR1C	UMSEL11	UMSEL10	UPM11	UPM10	USBS1	UCSZ11	UCSZ10	UCPOL1
(0xC9)	UCSR1B	RXCIE1	TXCIE1	UDRIE1	RXEN1	TXEN1	UCSZ12	RXB81	TXB81
(0xC8)	UCSR1A	RXC1	TXC1	UDRE1	FE1	DOR1	PE1	U2X1	MPCM1
(0xC7)	CLKSTA	–	–	–	–	–	–	RCON	EXTON
(0xC6)	CLKSEL1	RCCKSEL3	RCCKSEL2	RCCKSEL1	RCCKSEL0	EXCKSEL3	EXCKSEL2	EXCKSEL1	EXCKSEL0
(0xC5)	CLKSEL0	RCSUT1	RCSUT0	EXSUT1	EXSUT0	RCE	EXTE	–	CLKS
(0xC4)	TCCR4E	TLOCK4	ENHC4	OC4OE5	OC4OE4	OC4OE3	OC4OE2	OC4OE1	OC4OE0
(0xC3)	TCCR4D	FPIE4	FPEN4	FPNC4	FPES4	FPAC4	FPF4	WGM41	WGM40
(0xC2)	TCCR4C	COM4A1S	COM4A0S	COM4B1S	COM4B0S	COM4D1S	COM4D0S	FOC4D	PWM4D
(0xC1)	TCCR4B	PWM4X	PSR4	DTPS41	DTPS40	CS43	CS42	CS41	CS40
(0xC0)	TCCR4A	COM4A1	COM4A0	COM4B1	COM4B0	FOC4A	FOC4B	PWM4A	PWM4B
(0xBF)	TC4H						时钟 / 计数器 4 高字节		
(0xBE)	TCNT4	时钟 / 计数器 4 低字节							
(0xBD)	TWAMR	TWAM6	TWAM5	TWAM4	TWAM3	TWAM2	TWAM1	TWAM0	–
(0xBC)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
(0xBB)	TWDR	双线串口数据寄存器							
(0xBA)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
(0xB9)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0
(0xB8)	TWBR	双线串口位速率寄存器							
(0xB6)	Reserved	–	–	–	–	–	–	–	–
(0xB5)	Reserved	–	–	–	–	–	–	–	–
(0xB4)	Reserved	–	–	–	–	–	–	–	–
(0xB3)	Reserved	–	–	–	–	–	–	–	–
(0xB2)	Reserved	–	–	–	–	–	–	–	–
(0xB1)	Reserved	–	–	–	–	–	–	–	–
(0xB0)	Reserved	–	–	–	–	–	–	–	–
(0xAF)	Reserved	–	–	–	–	–	–	–	–
(0xAE)	Reserved	–	–	–	–	–	–	–	–
(0xAD)	Reserved	–	–	–	–	–	–	–	–
(0xAC)	Reserved	–	–	–	–	–	–	–	–
(0xB7)	Reserved	–	–	–	–	–	–	–	–
(0xAB)	Reserved	–	–	–	–	–	–	–	–
(0xAA)	Reserved	–	–	–	–	–	–	–	–
(0xA9)	Reserved	–	–	–	–	–	–	–	–

（续）

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
(0xA8)	Reserved	–	–	–	–	–	–	–	–
(0xA7)	Reserved	–	–	–	–	–	–	–	–
(0xA6)	Reserved	–	–	–	–	–	–	–	–
(0xA5)	Reserved	–	–	–	–	–	–	–	–
(0xA4)	Reserved	–	–	–	–	–	–	–	–
(0xA3)	Reserved	–	–	–	–	–	–	–	–
(0xA2)	Reserved	–	–	–	–	–	–	–	–
(0xA1)	Reserved	–	–	–	–	–	–	–	–
(0xA0)	Reserved	–	–	–	–	–	–	–	–
(0x9F)	Reserved	–	–	–	–	–	–	–	–
(0x9E)	Reserved	–	–	–	–	–	–	–	–
(0x9D)	OCR3CH	时钟 / 计数器 3：输出比较寄存器 C 高字节							
(0x9C)	OCR3CL	时钟 / 计数器 3：输出比较寄存器 C 低字节							
(0x9B)	OCR3BH	时钟 / 计数器 3：输出比较寄存器 B 高字节							
(0x9A)	OCR3BL	时钟 / 计数器 3：输出比较寄存器 B 低字节							
(0x99)	OCR3AH	时钟 / 计数器 3：输出比较寄存器 A 高字节							
(0x98)	OCR3AL	时钟 / 计数器 3：输出比较寄存器 A 低字节							
(0x97)	ICR3H	时钟 / 计数器 3：输入捕获寄存器高字节							
(0x96)	ICR3L	时钟 / 计数器 3：输入捕获寄存器低字节							
(0x95)	TCNT3H	时钟 / 计数器 3：计数器寄存器高字节							
(0x94)	TCNT3L	时钟 / 计数器 3：计数器寄存器低字节							
(0x93)	Reserved	–	–	–	–	–	–	–	–
(0x92)	TCCR3C	FOC3A	–	–	–	–	–	–	–
(0x91)	TCCR3B	ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30
(0x90)	TCCR3A	COM3A1	COM3A0	COM3B1	COM3B0	COM3C1	COM3C0	WGM31	WGM30
(0x8F)	Reserved	–	–	–	–	–	–	–	–
(0x8E)	Reserved	–	–	–	–	–	–	–	–
(0x8D)	OCR1CH	时钟 / 计数器 1：输出比较寄存器 C 高字节							
(0x8C)	OCR1CL	时钟 / 计数器 1：输出比较寄存器 C 低字节							
(0x8B)	OCR1BH	时钟 / 计数器 1：输出比较寄存器 B 高字节							
(0x8A)	OCR1BL	时钟 / 计数器 1：输出比较寄存器 B 低字节							
(0x89)	OCR1AH	时钟 / 计数器 1：输出比较寄存器 A 高字节							
(0x88)	OCR1AL	时钟 / 计数器 1：输出比较寄存器 A 低字节							
(0x87)	ICR1H	时钟 / 计数器 1：输入捕获寄存器高字节							
(0x86)	ICR1L	时钟 / 计数器 1：输入捕获寄存器低字节							
(0x85)	TCNT1H	时钟 / 计数器 1：计数器寄存器高字节							
(0x84)	TCNT1L	时钟 / 计数器 1：计数器寄存器低字节							
(0x83)	Reserved	–	–	–	–	–	–	–	–
(0x82)	TCCR1C	FOC1A	FOC1B	FOC1C	–	–	–	–	–
(0x81)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
(0x80)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10
(0x7F)	DIDR1	–	–	–	–	–	–	–	AIN0D
(0x7E)	DIDR0	ADC7D	ADC6D	ADC5D	ADC4D	–	–	ADC1D	ADC0D
(0x7D)	DIDR2	–	–	ADC13D	ADC12D	ADC11D	ADC10D	ADC9D	ADC8D

（续）

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
(0x7C)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
(0x7B)	ADCSRB	ADHSM	ACME	MUX5	–	ADTS3	ADTS2	ADTS1	ADTS0
(0x7A)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
(0x79)	ADCH	ADC 数据寄存器高字节							
(0x78)	ADCL	ADC 数据寄存器低字节							
(0x77)	Reserved	–	–	–	–	–	–	–	–
(0x76)	Reserved	–	–	–	–	–	–	–	–
(0x75)	Reserved	–	–	–	–	–	–	–	–
(0x74)	Reserved	–	–	–	–	–	–	–	–
(0x73)	Reserved	–	–	–	–	–	–	–	–
(0x72)	TIMSK4	OCIE4D	OCIE4A	OCIE4B	–	–	TOIE4	–	–
(0x71)	TIMSK3	–	–	ICIE3	–	OCIE3C	OCIE3B	OCIE3A	TOIE3
(0x70)	Reserved	–	–	–	–	–	–	–	–
(0x6F)	TIMSK1	–	–	ICIE1	–	OCIE1C	OCIE1B	OCIE1A	TOIE1
(0x6E)	TIMSK0	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0
(0x6D)	Reserved	–	–	–	–	–	–	–	–
(0x6C)	Reserved	–	–	–	–	–	–	–	–
(0x6B)	PCMSK0	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0
(0x6A)	EICRB	–	–	ISC61	ISC60	–	–	–	–
(0x69)	EICRA	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00
(0x68)	PCICR	–	–	–	–	–	–	–	PCIE0
(0x67)	RCCTRL	–	–	–	–	–	–	–	RCFREQ
(0x66)	OSCCAL	RC 振荡器校准寄存器							
(0x65)	PRR1	PRUSB	–	–	PRTIM4	PRTIM3	–	–	PRUSART1
(0x64)	PRR0	PRTWI	–	PRTIM0	–	PRTIM1	PRSPI	–	PRADC
(0x63)	Reserved	–	–	–	–	–	–	–	–
(0x62)	Reserved	–	–	–	–	–	–	–	–
(0x61)	CLKPR	CLKPCE	–	–	–	CLKPS3	CLKPS2	CLKPS1	CLKPS0
(0x60)	WDTCSR	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C
0x3E (0x5E)	SPH	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
0x3C (0x5C)	Reserved	–	–	–	–	–	–	–	–
0x3B (0x5B)	RAMPZ	–	–	–	–	–	–	RAMPZ1	RAMPZ0
0x3A (0x5A)	Reserved	–	–	–	–	–	–	–	–
0x39 (0x59)	Reserved	–	–	–	–	–	–	–	–
0x38 (0x58)	Reserved	–	–	–	–	–	–	–	–
0x37 (0x57)	SPMCSR	SPMIE	RWWSB	SIGRD	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
0x36 (0x56)	Reserved	–	–	–	–	–	–	–	–
0x35 (0x55)	MCUCR	JTD	–	–	PUD	–	–	IVSEL	IVCE
0x34 (0x54)	MCUSR	–	–	USBRF	JTRF	WDRF	BORF	EXTRF	PORF
0x33 (0x53)	SMCR	–	–	–	–	SM2	SM1	SM0	SE
0x32 (0x52)	PLLFRQ	PINMUX	PLLUSB	PLLTM1	PLLTM0	PDIV3	PDIV2	PDIV1	PDIV0
0x31 (0x51)	OCDR/ MONDR	OCDR7	OCDR6	OCDR5	OCDR4	OCDR3	OCDR2	OCDR1	OCDR0
		监视器数据寄存器							
0x30 (0x50)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
0x2F (0x4F)	Reserved	–	–	–	–	–	–	–	–
0x2E (0x4E)	SPDR	SPI 数据寄存器							
0x2D (0x4D)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X
0x2C (0x4C)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
0x2B (0x4B)	GPIOR2	通用 I/O 寄存器 2							
0x2A (0x4A)	GPIOR1	通用 I/O 寄存器 1							
0x29 (0x49)	PLLCSR	–	–	–	PINDIV	–	–	PLLE	PLOCK
0x28 (0x48)	OCROB	时钟 / 计数器 0 输出比较寄存器 B							
0x27 (0x47)	OCROA	时钟 / 计数器 0 输出比较寄存器 A							
0x26 (0x46)	TCNT0	时钟 / 计数器 0（8 位）							
0x25 (0x45)	TCCR0B	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00
0x24 (0x44)	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00
0x23 (0x43)	GTCCR	TSM	–	–	–	–	–	PSRASY	PSRSYNC
0x22 (0x42)	EEARH	–	–	–	–	EEPROM Address Register High Byte			
0x21 (0x41)	EEARL	EEPROM 地址寄存器低字节							
0x20 (0x40)	EEDR	EEPROM 数据寄存器							
0x1F (0x3F)	EECR	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE
0x1E (0x3E)	GPIOR0	通用 I/O 寄存器 0							
0x1D (0x3D)	EIMSK	–	INT6	–	–	INT3	INT2	INT1	INT0

（续）

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
0x1C (0x3C)	EIFR	–	INTF6	–	–	INTF3	INTF2	INTF1	INTF0
0x1B (0x3B)	PCIFR	–	–	–	–	–	–	–	PCIF0
0x1A (0x3A)	Reserved	–	–	–	–	–	–	–	–
0x19 (0x39)	TIFR4	OCF4D	OCF4A	OCF4B	–	–	TOV4	–	–
0x18 (0x38)	TIFR3	–	–	ICF3	–	OCF3C	OCF3B	OCF3A	TOV3
0x17 (0x37)	Reserved	–	–	–	–	–	–	–	–
0x16 (0x36)	TIFR1	–	–	ICF1	–	OCF1C	OCF1B	OCF1A	TOV1
0x15 (0x35)	TIFR0	–	–	–	–	–	OCF0B	OCF0A	TOV0
0x14 (0x34)	Reserved	–	–	–	–	–	–	–	–
0x13 (0x33)	Reserved	–	–	–	–	–	–	–	–
0x12 (0x32)	Reserved	–	–	–	–	–	–	–	–
0x11 (0x31)	PORTF	PORTF7	PORTF6	PORTF5	PORTF4	–	–	PORTF1	PORTF0
0x10 (0x30)	DDRF	DDF7	DDF6	DDF5	DDF4	–	–	DDF1	DDF0
0x0F (0x2F)	PINF	PINF7	PINF6	PINF5	PINF4	–	–	PINF1	PINF0
0x0E (0x2E)	PORTE	–	PORTE6	–	–	–	PORTE2	–	–
0x0D (0x2D)	DDRE	–	DDE6	–	–	–	DDE2	–	–
0x0C (0x2C)	PINE	–	PINE6	–	–	–	PINE2	–	–
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
0x08 (0x28)	PORTC	PORTC7	PORTC6	–	–	–	–	–	–
0x07 (0x27)	DDRC	DDC7	DDC6	–	–	–	–	–	–
0x06 (0x26)	PINC	PINC7	PINC6	–	–	–	–	–	–
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0

（续）

地址	名称	位7	位6	位5	位4	位3	位2	位1	位0
0x02 (0x22)	Reserved	–	–	–	–	–	–	–	–
0x01 (0x21)	Reserved	–	–	–	–	–	–	–	–
0x00 (0x20)	Reserved	–	–	–	–	–	–	–	–

附录C

Arduino与兼容产品厂商

请注意，此处列出的厂商只是一种购买渠道，供各位购买相关产品时作参考，并非有意为它们进行广告宣传（当然 Arduino 官方除外）。

C.1 Arduino产品

购买 Arduino 产品的主要渠道当然是 Arduino 官方公司。在其官方网站（Arduino.cc）上，你可以看到所有在售的产品。此外，还有许多经销商也在销售 Arduino 开发板、扩展板以及相关配件。

C.1.1 硬件可兼容开发板与扩展板

名称	URL	名称	URL
Adafruit	www.adafruit.com	ITEAD Studio	store.iteadstudio.com
Arduino	store.arduino.cc	Macetech	www.macetech.com/store/
Arduino Lab	www.arduinolab.us	Mayhew Labs	mayhewlabs.com
Circuit@tHome	www.circuitsathome.com	Nootropic Design	nootropicdesign.com
CuteDigi	store.cutedigi.com	Numato	numato.com
DFRobot	www.dfrobot.com	RobotShop	www.robotshop.com
DealeXtreme (DX)	www.dx.com	Rugged Circuits	www.ruggedcircuits.com
Elecfreaks	www.elecfreaks.com	SainSmart	www.sainsmart.com
Elechouse	www.elechouse.com	Seeed Studio	www.seeedstudio.com
excamera	www.excamera.com	SparkFun	www.sparkfun.com
Iowa Scaled Engineering	www.iascaled.com	Tindie	www.tindie.com
iMall	imall.itead.cc	Tronixlabs	tronixlabs.com

C.1.2 软件可兼容开发板

名称	URL
Adafruit	www.adafruit.com
Circuit Monkey	www.circuitmonkey.com
BitWizard	www.bitwizard.nl

C.1.3 传感器、扩展板、模块

名称	URL	名称	URL
Adafruit	www.adafruit.com	Seeed Studio	www.seeedstudio.com
CuteDigi	store.cutedigi.com	TinyCircuits	www.tiny-circuits.com
DealExtreme (DX)	www.dx.com	Trossen Robotics	www.trossenrobotics.com
KEYES	en.keyes-robot.com	Vetco	www.vetco.net

C.2 电子软件

C.2.1 开源电路图绘制工具

名称	URL
ITECAD	http://www.itecad.it/en/index.html
Oregano	https://github.com/marc-lorber/oregano
Open Schematic Capture (OSC)	http://openschcapt.sourceforge.net
TinyCAD	http://sourceforge.net/apps/mediawiki/tinycad
XCircuit	http://opencircuitdesign.com/xcircuit

C.2.2 CAE软件工具

名称	描述	URL
DesignSpark	免费，非开源	http://www.rs-online.com/designspark/electronics/
Eagle	免费，非开源	http://www.cadsoftusa.com
Fritzing	免费 CAE 工具	http://fritzing.org/home
gEDA	开源 CAE 工具	http://www.geda-project.org
KiCad	开源 CAE 工具	http://www.kicad-pcb.org

C.2.3 PCB布局设计工具

名称	描述	URL
FreePCB	只在 Windows 系统下可用	http://www.freepcb.com
FreeRouting	基于 Web 的自动布线器	http://www.freerouting.net
PCB	Linux 开源工具	http://sourceforge.net/projects/pcb/

C.3 硬件、元件与工具

C.3.1 电子元件制造商

名称	URL	名称	URL
Allegro	http://www.allegromicro.com	Micrel	http://www.micrel.com
Analog Devices	http://www.analog.com	Microchip	http://www.microchip.com
ASIX	http://www.asix.com.tw	NXP	http://www.nxp.com
Atmel	http://www.atmel.com	ON Semiconductor	http://www.onsemi.com
Bluegiga	http://www.bluegiga.com	Panasonic	http://www.panasonic.com
Cypress	http://www.cypress.com	Silicon Labs	http://www.silabs.com
Digi International	http://www.digi.com	STMicrotechnology	http://www.st.com
Fairchild	http://www.fairchildsemi.com	Texas Instruments	http://www.ti.com
FTDI	http://www.ftdichip.com	WIZnet	http://www.wiznet.co.kr
Linear Technology	http://www.linear.com	Zilog	http://www.zilog.com

C.3.2 电子元件分销商（USA）

名称	URL
Allied	http://www.alliedelec.com
Digi-Key	http://www.digikey.com
Jameco	http://www.jameco.com
Mouser	http://www.mouser.com
Newark/Element14	http://www.newark.com
Parts Express	http://www.parts-express.com
State	http://www.potentiometer.com

C.3.3 折扣与多余电子元件

名称	URL
All Electronics	http://www.allelectronics.com
Alltronics	http://www.alltronics.com
American Science & Surplus	http://www.sciplus.com
BGMicro	http://www.bgmicro.com
Electronic Surplus	http://www.electronicsurplus.com
Electronic Goldmine	http://www.goldmine-elec-products.com

C.3.4　机械配件与硬件（螺丝钉、螺母、螺栓）

名称	URL	名称	URL
All Electronics	http://www.allelectronics.com	McMaster-Carr	http://www.mcmaster.com
Alltronics	http://www.alltronics.com	Micro Fasteners	http://www.microfasteners.com
Bolt Depot	http://www.boltdepot.com	SDP/SI	http://www.sdp-si.com
Fastenal	http://www.fastenal.com	WM Berg	http://www.wmberg.com

C.3.5　外壳与机箱

名称	URL	名称	URL
Bud Industries	http://www.budind.com	LMB Heeger	http://www.lmbheeger.com
Context Engineering	http://contextengineering.com/index.html	METCASE/OKW Enclosures	http://www.metcaseusa.com
ELMA	http://www.elma.com	Polycase	http://www.polycase.com
Hammond Manufacturing	http://www.hammondmfg.com/index.htm	Serpac	http://www.serpac.com
iProjectBox	http://www.iprojectbox.com	TEKO Enclosures	http://www.tekoenclosures.com/en/home

C.3.6　工具

名称	URL	名称	URL
Adafruit	http://www.adafruit.com	Maker Shed	http://www.makershed.com
Apex Tool Group	http://www.apexhandtools.com	MCM Electronics	http://www.mcmelectronics.com
CKB Products	http://www.ckbproducts.com	SainSmart	http://www.sainsmart.com
Circuit Specialists	http://www.circuitspecialists.com	SparkFun	http://www.sparkfun.com
Electronic Goldmine	http://www.goldmine-elec-products.com	Stanley	http://www.stanleysupplyservices.com
Harbor Freight Tools	http://www.harborfreight.com	Velleman	http://www.vellemanusa.com

C.3.7　测试设备

名称	URL	名称	URL
Adafruit	http://www.adafruit.com	SparkFun	http://www.sparkfun.com
Electronic Goldmine	http://www.goldmine-elec-products.com	Surplus Shed	http://www.surplusshed.com
MCM Electronics	http://www.mcmelectronics.com	Velleman	http://www.vellemanusa.com

C.4 印制电路板供应商与制造商

大多数大的电子元件分销商都会销售诸如蚀刻剂以及单双面覆铜 PCB（带有光致抗蚀剂）。如果不想接触这些化学制剂及相关工艺，可以考虑委托商业原型 PCB 制造商为你制作 PCB。

C.4.1 原型与快速周转制造商

名称	URL
Advanced Circuits	http://www.4pcb.com
ExpressPCB	http://www.expresspcb.com
Gold Phoenix PCB Co.	http://www.goldphoenixpcb.com
Sunstone Circuits	http://www.sunstone.com/
Sierra Circuits	https://www.protoexpress.com

C.4.2 PCB工具厂商

厂商名称	URL	产品
Jameco Electronics	http://www.jameco.com	传统酸蚀刻及物料
Think & Tinker, Ltd.	http://www.thinktink.com	制造 PCB 的各种物料
Vetco Electronics	http://www.vetco.net	传统酸蚀刻工具

C.5 其他购买渠道

通过简单的搜索，你可以在网上搜到大量厂商，上面列出的只是其中一部分。此外，还有 Amazon、Mouser 网上经销商，通过它们可以买到各种 Arduino 以及兼容 Arduino 的产品。eBay 也是一个不错的地方，上面有很多亚洲厂商，从他们那里你可以买到很多便宜的电子元件（一定要注意看其评级，但一般都很高）。最后不要忘了使用谷歌搜索，在搜索框中输入 Arduino 会返回大约 4000 万条搜索结果（写作本书时）。

附录D

推荐阅读

对于技术类图书作者（包括我），Arduino 是个很热门的主题。市面上有大量讲解 Arduino 的图书，其中一些介绍 Arduino 在特定范围内的应用，其他更多的是按照一系列的项目进行介绍。除了专门讲解 Arduino 的图书之外，还有其他一些图书也值得阅读，它们所讲内容主要涉及 AVR 微控制器、C 与 C++ 编程、常见电子元件、接口、仪器、印刷电路板。

D.1 Arduino

- 《爱上 Arduino》
- *Atmospheric Monitoring with Arduino*, Patrick Di Justo & Emily Gertz, (Maker Media). 2013. ISBN 978-1449338145
- *Environmental Monitoring with Arduino*, Emily Gertz & Patrick Di Justo, (Maker Media). 2012. ISBN 978-1449310561
- 《Arduino 编程从零开始》
- 《Arduino 实战入门手册：智能硬件制作项目大全》

D.2 AVR

- *Some Assembly Required: Assembly Language Programming with the AVR Microcontroller*, Timothy Margush, CRC Press. 2011. ISBN 978-1439820643
- *Make: AVR Programming*, Elliot Williams, Maker Media. 2014. ISBN 978-1449355784

D.3 C与C++编程

- 《C 程序设计语言》
- 《C 语言程序设计：现代方法》

- 《C++ Primer》
- 《C++ Primer Plus》

D.4 常见电子元器件

- *Data Conversion Handbook*, Analog Devices, Newnes. 2004. ISBN 978-0750678414
- *The 555 Timer Applications Sourcebook*, Howard Berlin, Howard W. Sams. 1976. ISBN 978-0672215381
- *The Electrical Engineering Handbook*, Richard Dorf (Ed.), CRC Press LLC. 1997. ISBN 978-0849385741
- *Electronics*, 2nd Edition, Allan Hambley, Prentice Hall, 1999. ISBN 978-0136919827
- *The Art of Electronics*, 2nd Edition, Paul Horowitz & Winfield Hill, Cambridge University Press. 1989. ISBN 978-0521370950
- 《电子工程师必读：元器件与技术》
- *IC Op-Amp Cookbook*, Walter G. Jung, Howard W. Sams. 1986. ISBN 978-0672224534
- *Contemporary Logic Design*, 2nd Edition, Randy Katz, Prentice Hall. 2004. ISBN 978-0201308570
- *Digital Electronics: A Practical Approach*, William Kleitz, Regents/Prentice Hall. 1993. ISBN 978-0132102870
- 《爱上电子学：创客的趣味电子实验》
- *Electronic Filter Design Handbook*, 4th Edition, Arthur Williams & Fred Taylor, McGraw-Hill. 2006. ISBN 978-0071471718

D.5 接口

- *Parallel Port Complete*, Jan Axelson, Lakeview Research LLC. 2000. ISBN 978-0965081917
- *Serial Port Complete*, Jan Axelson, Lakeview Research LLC. 2007. ISBN 978-1931448062
- *USB Complete*, Jan Axelson, Lakeview Research LLC. 2007. ISBN 978-1931448086
- *Essentials of Short-Range Wireless*, Nick Hunn, Cambridge University Press. 2010. ISBN 978-0521760690
- *USB: The Universal Serial Bus*, Benjamin Lunt, CreateSpace. 2012. ISBN 978-1468151985
- 《以太网权威指南（第 2 版）》

D.6 仪器

- *Real World Instrumentation with Python*, J. M. Hughes, O'Reilly. 2010. ISBN 978-0596809560

D.7 印制电路板

- *Making Printed Circuit Boards*, Jan Axelson, Tab Books. 1993. ISBN 978-0830639519
- *Fritzing for Inventors*, Simon Monk, McGraw-Hill. 2016. ISBN 978-0071844635
- *Make Your Own PCBs with Eagle*, Simon Monk, McGraw-Hill. 2014. ISBN 978-0071819251
- 《高质量 PCB 设计入门》

附录E

Arduino与AVR软件开发工具

本书主要讲解 Arduino IDE 与 AVR-GCC 工具链，但它们不是唯一可用的工具。此外还有许多不同工具可以用于汇编、编译、链接，以及将可执行代码加载到 AVR MCU 中。这些工具有些是开源的，有些是商业付费的，而有些工具比其他工具功能更强大、更易用。

E.1 编译器/汇编器

- **Windows 系统下的 Atmel AVR 工具链**（http://bit.ly/atmel-avr）
  开源工具套装，包括一个汇编器，移植到 Windows。包含 GNU 许可证软件与 Atmel 开发的工具。
- **AVR-GCC**（http://www.nongnu.org/avr-libc）
  包含一整套工具链组件，将 C 或 C++ 源码交叉编译为 AVR 可执行代码。更多内容参见第 6 章。
- **SDCC**（http://sdcc.sourceforge.net）
  一个开源 ANSI C 编译器，适用于各种微控制器。
- **WinAVR**（http://winavr.sourceforge.net）
  该开源工具来自 AVR-GCC 工具链，专为 Windows 环境量身打造。更多内容参见第 6 章。

E.2 集成开发环境

- **Arduino IDE**（https://www.arduino.cc）
  Arduino 官方专为 Arduino 硬件编写的集成开发环境，可以运行在 Windows、Linux、Mac OS X 平台下，开源、免费。更多内容参见第 5 章。

- **Atmel Studio 7**（http://bit.ly/atmel-studio-7）
  该 IDE 集成了 C/C++ 编译器，提供免费下载，只有针对 Windows 的版本（Windows7 或更高）。更多内容参见第 6 章。
- **Eclipse 插件**（http://bit.ly/avr-eclipse）
  面向 AVR 的 Eclipse 插件。Eclipse（https://eclipse.org）是一个基于 Java 的开源集成开发环境，可以运行在 Windows、Linux 与 Mac OS X 系统下。
- **IAR Embedded Workbench**（http://bit.ly/iar-workbench）
  高集成的专用工具套装。用户需要付费购买使用许可，免费试用 30 天，仅有 Windows 版本。
- **MikroElektronika mikroC**（http://www.mikroe.com/mikroc/avr）
  带有 IDE 的商业 ANSI C 编译器。单用户版本价格为 249 美元，适用于 Windows XP 或更高版本。
- **ImageCraft JumpStart**（https://www.imagecraft.com/devtools_AVR.html）
  商业 ANSI C 编译器，部分功能基于 GPL 开源软件实现。Windows 标准版为 249 美元，有可用的授权软件保护器。
- **Rowley CrossWorks**（http://www.rowley.co.uk/avr）
  带有 IDE 的商业多平台 ANSI C 编译器，授权价格为 150~2250 美元，具体取决于产品用途，可以运行在 Windows、Mac OS X 与 Linux 系统下。

E.3 编程工具

- **PlatformIO**（http://platformio.org）
  基于命令行的 AVR-GCC 工具链接口，可运行在 Windows、Linux 与 Mac OS X 系统下。更多内容参见第 6 章。
- **Ino**（http://inotool.org）
  基于命令行的 AVR-GCC 工具链接口，可运行在 Linux 与 Mac OS X 系统下。更多内容参见第 6 章。

E.4 模拟器

- **AMC VMLAB**（http://www.amctools.com/vmlab.htm）
  Windows 系统下的免费图形 AVR 模拟器。
- **GNU AVR Simulator**（http://sourceforge.net/projects/avr）
  开源 AVR 模拟器，带有基于 Motif 的图形界面，可运行在 Linux 与 Unix 下。
- **Labcenter Proteus**（http://www.labcenter.com/products/vsm/avr.cfm）
  基于 Novel 电路图的 AVR 模拟器，全图形界面。授权价格 248 美元起，仅有 Windows 版本。

- **OshonSoft AVR Simulator**（http://www.oshonsoft.com/avr.html）
  带有可选附加模块的图形 AVR 模拟器，个人授权价格为 32 美元。
- **SimulAVR**（http://www.nongnu.org/simulavr）
  基于命令行的开源 AVR 模拟器，运行在 Linux/Unix 系统下。

关于作者

John M. Hughes

嵌入式系统工程师，在电子学、嵌入式系统及软件、航天系统和科学应用开发等领域拥有30余年的从业经验。曾负责为凤凰号火星探测器开发表面成像软件。他所在的一个小组还开发了新型合成外差激光干涉仪，用于校正韦伯太空望远镜镜片的位置控制。另著有《电子工程师必读：元器件与技术》，为没有电子学专业背景的爱好者和创客系统介绍电子学中的硬件、元件、工具和技术。

关于封面

本书的封面图是一只玩具鸭。这种拉着走的木制玩具是乐高集团首批产品之一，于1935年推出，最初由公司创始人——丹麦木匠 Ole Kirk Christiansen 制作。Christiansen 先生从20世纪30年代开始生产木制玩具，当时丹麦大萧条刚刚开始。

乐高鸭有一个长方形底座，并装有可转动的轮子。用绳子拉着它走时，鸭子的嘴巴会一张一合。最初鸭子身体被涂成红色，鸭头、尾巴、翅膀则为黑色，后来推出的玩具鸭颜色则更加多样化。

20世纪60年代，乐高停止生产木制玩具，导致这种标志性的玩具鸭特别稀少。2011年，乐高公司推出一款新型玩具鸭，它由我们今天熟知的乐高积木搭建而成。

O’Reilly 图书封面上的很多动物都是濒危动物，它们对世界都很重要。要想详细了解如何帮助这些动物，请访问 animals.oreilly.com。

本书封面图片选自 Ronald Barlow 的 *The Great American Antique Toy Bazaar* 一书。

O'REILLY®

Arduino技术指南

作为成功的开源硬件平台，Arduino不但让普通人实现了工程师梦想，也广泛应用于机器人、环境监测传感器、卫星等大大小小的正式项目。

本书针对想了解Arduino细节、追求个性和新鲜事物的爱好者，详尽介绍了使用Arduino开发板与扩展板的过程中应该了解的技术细节、注意事项，并传达了一些非常重要的设计思想，旨在帮助读者实现创客梦想。

- **Arduino开发板的物理特性与接口功能**
- **Arduino使用的各种AVR微控制器**
- **Arduino特有的编程环境，包括“程序”**
- **各种扩展板，如闪存、以太网、蓝牙、ZigBee**
- **可与Arduino一起工作的传感器、继电器模块、小键盘等附加组件**
- **从零开始创建自定义扩展板**
- **介绍如何分析各种设计问题、定义实体与可测试需求，确保开发成功**

John M. Hughes，嵌入式系统工程师，在电子学、嵌入式系统及软件、航天系统和科学应用开发等领域拥有30余年从业经验。曾负责为“凤凰号”火星探测器开发表面成像软件。他所在的一个小组还开发了新型合成外差激光干涉仪，用于校正韦伯太空望远镜镜片的位置控制。另著有《电子工程师必读：元器件与技术》，为没有电子学专业背景的爱好者和创客系统介绍电子学中的硬件、元件、工具和技术。

“本书从工程师视角讲解了Arduino平台的各种细节。如果你想进一步了解Arduino的内部工作原理，而不止于知道如何使用，那么本书将是你的不二之选。”

——Simon Monk

《Arduino编程指南》作者

SOFTWARE ENGINEERING / ROBOTICS

封面设计：Randy Comer　张健

图灵社区：iTuring.cn
热线：(010)51095186转600

分类建议　计算机 / Arduino

人民邮电出版社网址：www.ptpress.com.cn

O'Reilly Media, Inc.授权人民邮电出版社出版

ISBN 978-7-115-47105-5

ISBN 978-7-115-47105-5
定价：129.00元

看完了

如果您对本书内容有疑问，可发邮件至 contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱：ebook@turingbook.com。

在这可以找到我们：

微博 @图灵教育：好书、活动每日播报
微博 @图灵社区：电子书和好文章的消息
微博 @图灵新知：图灵教育的科普小组
微信 图灵访谈：ituring_interview，讲述码农精彩人生
微信 图灵教育：turingbooks